Notes on the Parallel Decomposition Theory of Finite State Machines

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computer Science at the University of Canterbury

Kahn Mason

July 14, 1998

This thesis is dedicated to my grandfather, Lawrence Chaston (26/6/1918 - 22/6/1998).

Thank you for everything.

Contents

List of Figures

Abstract

We extend existing theory for the parallel decomposition of finite machines (finite automata) to ω-machines and timed machines. The focus for all three is the existence of a structural relationship between the decomposition and the original machine. This is defined in terms of suitable homomorphisms. The homomorphisms also yield inuitively obvious relationships between the languages of accepted words.

The theory of decomposition by state partitions obtaining quotient machines is known for finite machines [Hol82, Shi87]. The extensions to ω-machines is straightforward with a suitable choice of acceptance criteria. Muller acceptance criteria [Tho90] seem natural and are used here. A suitable partial order on the partitions leads to a lattice, the minimal elements of which are the natural starting points in locating decompositions.

The extension to timed machines [AD94] is not as straightforward. As anticipated clock resetting and constraints prevent a straightforward state based generalisation. Suitable partitions of both states and clocks are required to generate quotient machines. Once again, a suitable partial order leads to a regarding a lattice, the minimal elements of which are natural starting points. The members of the lattice are now pairs of state partitions with clock subsets.

Each of the theories is developed alongside a worked example illustrating how the theory is applied. Discussion of the results, their potential applications and areas of concern is interleaved with the results, and is summarised at the end.

Acknowledgements

I have been told that writing a thesis is as close as a male can come to bearing a child. If that is the case then my supervisor, Padmanabhan Krishnan, deserves much credit in his role as midwife for it has at times been a challenging birth. My sincerest thanks also to John Hannah for his unheralded but unfailing support during the last year and a half. The G B Battersby-Trimble Scholarship in Computer Science helped fund this, and made my life considerably easier than would otherwise have been. All the people helping me over the past year, and their have been many, deserve recognition for their part in this thesis but I dare not for fear of missing someone. The birth analogy is courtesy of Jane Clucas.

Chapter 1

Introduction

1.1 Why do we decompose?

1.1.1 Decomposition in general

A fundamental problem in almost every area of human thought and study is how we can view things as collectives, rather than as individuals, and the relationships that exist between individuals and collectives. The relations between individual behaviour and global behaviour is increasingly under the spotlight — certainly in computer science, where parallel computing and hardware design are seen as a much more practical and powerful approach to problem solving than the serial paradigm.

But it is not only the drive of the computer scientist or electrical engineer that encourages us to study decomposition theory. Connectionism is an increasingly popular account for human cognition in areas of philosophy and psychology. Physics, by its very nature, is concerned with the accounting for of global properties from local ones. In Economics we attempt to use our knowledge of individuals to predict and understand the marketplace. Indeed, there are many more such examples, and the study of part-whole relationships is both a fundamental and a very important one.

Science seeks to note the patterns, or structures present in the world around us, and thus provide as concise a description of the world as possible. Decomposition is a very similar concept—the regularities of a large structure, or machine, are taken advantage of to provide a smaller description of the structure. As is suggested by the name, the usual method of describing a large machine is in terms of smaller components linked together in some manner.

There is often a loss of fine detail in the treatment of a large machine as components, and we want this ignored detail to be irrelevant to the problem at hand. A good example of this is in Fourier Analysis where a complicated signal is "broken up" into harmonics, and the lower harmonics are used as an approximation to the signal. Here the signal is the object to be decomposed, each harmonic is a component, and combining components corresponds to superimposing the signals. The effectiveness of this process depends critically on the fact that the higher harmonics contribute only a small amount to the signal, an assumption which holds in almost every signal application.

All this puts decomposition theory on a very high pedestal, but one upon which it rests very comfortably. However, the study of decomposition in general is not our focus. Our aim is to consider some of the more common abstract machines in use today, and take advantage of the regularities that many of the possess. Thus this study is motivated by experience as opposed to being purely hypothetical.

1.1.2 Specifics

We focus in this thesis on a particular type of decomposition, namely that of a parallel decomposition, and restrict our attention to a certain class of abstract machines, each of which has a finite specification. To be more precise the decompositions are of the lock-step parallel variety, meaning that each component processes at exactly the same rate, and each component processes each input from the environment. The sorts of machines we are looking at are all finite machines, ω-machines, or timed machines and we assume that their structural presentation follows particular forms. These are presented in Definition 2.22, Definition 2.26, and Definition 2.32 respectively.

As discussed earlier, these sorts of decompositions take account of the regularities that are present in many of these sorts of machines. We consider the three classes of machines mentioned in the previous paragraph for differing reasons. We look at finite machines because the theory has already being developed for them, and we use this existing theory as a starting point. We look at ω-machines because their structural presentation is so similar to that of finite machines, and they provide a very natural extension to deal with non-terminating computation. The main thrust of the work is the extension to timed machines, because the introduction of time into the specifications is an area of current interest, and the specifications as given are very commonly used. The introduction of time allows for much more complicated systems to be represented. Coupling this with some decomposition results has the potential to increase vastly the range of systems that it is feasible to study using the general techniques of abstract machine theory.

The use of finite machines is prevalent in many areas of Computer Science and Electrical Engineering. Model checking, for example of programs or circuits, is a particularly important area [Eme90, Var96]. The predominant problem encountered in this application is that the size of the finite machine grows very

fast as the size of the problem, be it code length for programs, or the number of logic gates for a circuit, increases. In model checking, binary decision diagrams [Bry86, And96] are a common representation tool which allow for the practical representation of finite machines of state sizes up to approximately 10^{20} on current systems [BCMH92].

Finite machines are also used in controller systems [EA95] whereby a "plant" modelled by a finite machine is influenced by a "controller" to ensure that it satisfies certain properties [EA95]. Describing this as the controller in parallel with the plant is very common. Normally the controller acts to restrict the behaviour of the plant. This is easily achieved by considering the plant to be operating in parallel with the controller. This means that acceptable sequences for the system consisting of both the plant and the controller must be acceptable to both the plant and the controller. One particularly important subfield is in controller synthesis, where the description of a plant is given along with a property that it is desired for the system to hold. The controller is then generated to ensure that the system does indeed satisfy the property in question, which is typically referred to as a safety property deriving from the fact that the system is being restricted so that unwanted events do not happen. These sorts of properties do not ensure that desired events do happen. The area dealing with plants and controllers is referred to as hybrid, or integrated, systems theory and the problem of controller synthesis is a well established one [ZM95].

Decompositions should allow for smaller representations of both plants and controllers, thus making it more practical to deal with more complicated systems. Specific issues relating to each of the aforementioned applications are likely to arise.

1.2 How do we decompose?

1.2.1 The problem

We often deal with large (monolithic) machines of of the types we are considering here, and we we want to be able to represent them as a parallel machine. That is, we want to determine if it has parallel components, and if it does then to find them. Consider the first of these two problems. We wish to know if a large, given, machine can be represented as a parallel one. The trivial criterion is that the machine can be represented as a parallel one precisely when smaller machines can be found whose parallel composition represents the larger machine. From an implementation point of view this criterion is not so helpful because it suggests no other method for checking existence of a parallel decomposition than trying all collections of smaller machines, and for each collection seeing if their composition represents the larger machine. With this in mind, the problem is to find what regularities must be present in the larger machine in order that it can be represented as a parallel one. But we wish to know more than

the existence of a parallel decomposition, we also wish to find the smaller machines that realise the decomposition. Thus, an additional problem is to actually find the components that together represent the larger machine. The goal is thus to turn the problem of finding parallel decompositions into a test of the algebraic structure of the machine. This will hopefully mean that automating the process is much more feasible. The approach here is to extend the theory of [Shi87] from finite automata to ω- and timed automata.

1.2.2 The organisation of this thesis

Chapter ?? introduces and discusses most of the terminology used for the duration of the thesis. Much is standard, but because this work involves bringing together ideas from different fields, we discuss the notation used so that it can reach a wider audience. Not all of the discussion is directly pertinent, but is present to provide a context, and also to help develop the intuitions necessary to manipulate the concepts involved.

Following this discussion there is a section on each of the three types of abstract machines we will considering. Each section is largely a review of a standard introductory work; [HU79] for finite machines, [Tho90] for ω-machines, and [AD94] for timed machines. We refer the reader to the appropriate papers for more complete discussions. In each of these papers, the abstract machines are referred to as automata, and in machine theory the two words are synonyms.

Chapter 3 reviews existing theory for the parallel decomposition of finite machines [Shi87], and concurrently extends this theory to ω-machines. The theory of [Shi87] is an instance of the more general Khron-Rhodes decomposition theory presented in [Hol82], but we do not try to extend the full generality of the Khron-Rhodes decomposition, and restrict our attention to parallel decompositions. We find for parallel decompositions that the theory carries over almost word for word, which is the reason the two theories are presented here simultaneously. The format, both of the chapter and the results, is incremental in nature with a mind to later extensions to timed machines.

The first task is to describe the parallel composition of machines as a machine of the same type. Each of the classes of machines considered here is closed under the appropriate definition of parallel composition. Our motivation for the definitions is that each component will process at the same rate and independently, a transition only being performed if each component can make a transition. We also have a parallel machine accepting a word precisely when it is each component does.

After this we formalise what it means to represent another machine. We choose suitable concepts of homomorphism, or structural inclusion. Isomorphism is too restrictive, and in practice some global synchronisation is used to achieve this. Homomorphism is a sufficiently weak concept to enable us to

decompose many realistic machines.

The development of these concepts enables a formal definition of a parallel decomposition, where everything is described in terms of the structural relationships between the machines. We concentrate attention on the two component case to avoid the manipulation of complicated notation. It eventuates that the two component case is sufficient so there is no loss of generality. The result showing this is one of the later ones in the chapter because it is a simple corollary of some of the more developed concepts.

The conversion of the problem of finding a decomposition to the recognition of a structural property begins with finding necessary properties of the large machine. This leads to the eventual development of a form of quotient machine induced by state partitions. Requiring the quotient machines to be deterministic forces these partitions to have the substitution property, and partitions with this property are called SP partitions. The necessary conditions for a parallel decomposition are then phrased in terms of suitable SP partitions, most importantly that they have to be orthogonal.

Throughout the development of the necessary conditions a simple example was used as motivation, and the development of the sufficient conditions uses the same example. The presence of suitable partitions generates quotient machines which serve to realise a parallel decomposition. Thus the sufficient conditions match the necessary ones, completing the conversion of the problem of finding decompositions in the two component case. A natural lattice on the set of SP partitions of a machine leads to the result that the two component case is sufficient completing the conversion for the general case.

The last part of Chapter 3 is devoted to the application of the theory to the problem of decomposing a simple buffer of size two. Although this is a small example, it illustrates many of the issues that arise in implementation. An informal procedure for locating suitable partitions is used, and some problems are mentioned.

Chapter 4 mimics Chapter 3, both in form and content except that it deals with timed machines. Thus there is a developed example interleaved with the development, first of concepts of parallel composition and homomorphism, and then of both necessary and sufficient structural properties to ensure the existence of a parallel decomposition.

Once more the results proceed via the development of quotient machines. However, the introduction of clocks means that the concept of a quotient machine incorporates the possibility of a restricted clock set in addition to a state partition, and determinism requires that the pairing of the state partition and restricted clock set has a particular property, here called admissibility. Suitable admissible pairs generate quotient machines realising parallel decompositions as SP partitions did for finite and ω-machines. The clock introduction means that suitable admissible pairs require considering both clock splits and orthogonal state partitions together. On the face of it, this seems a stricter condition than that of SP partitions, but

this may be misleading. Again, the chapter closes with an illustrative example and an informal procedure for generating suitable admissible pairs. It appears that concentrating on the clocks is a natural starting point, contrasting the finite and ω-machine cases.

The thesis closes with a brief summary, and a discussion of the issues that are likely to arise during implementation, as well as possible extensions. Mention is also made of the areas where this work is most likely to be of benefit. These are topics of further research.

Chapter 2

Preliminaries

This chapter introduces some terminology and discusses the assorted machines that will be considered in this thesis. We cover the formal definitions of machines, some results describing links between machines and formal languages, and give some examples. When we have transcribed a result, we attribute it immediately after its title, in which case proofs are presented only when the results will be generalised later, or we consider the proof itself illuminating. Results derived here are left unattributed.

2.1 Some Notational Conventions and Terminology

We introduce here a lot of the notation that is used in the remainder of the thesis. Much of the notation is standard, but is presented here for those unfamiliar with it. Some of the notation, however, is non-standard, and we recommend that the reader at least skim this chapter.

2.1.1 Sets

Throughout, we tend to use lower case Roman letters for elements of sets, upper case Roman letters for sets, and scripted upper case letters for collections of sets. This does get a little distorted at times because the elements of a set may themselves be sets. For a readable discussion on set theory we refer the reader to [Hal60].

If A and B are two sets, then we denote their set difference to be

$$A - B \triangleq \{x \mid x \in A, x \notin B\};$$

their product to be

$$A \times B \triangleq \{(a,b) \mid a \in A, b \in B\}$$

their symmetric difference to be

$$A \Delta B \triangleq (A \cup B) - (A \cap B);$$

and their disjoint union to be

$$A \uplus B \triangleq (A \times \{0\}) \cup (B \times \{1\})$$

We use both $\cup$ and $+$ for union. We let 2^X stand for the power set of X. There is a natural isomorphism between the two, so that this presents no problem. For those readers unfamiliar with the notation, A^B denotes the set of all functions from B to A, and 2 in this context (when regarded as a set) stands for the set $\{0,1\}$. Thus 2^X is the set of all functions from X into $\{0,1\}$, and has cardinality $2^{|X|}$.

If $X \subseteq A$, and $b \in B$, then we define $(X,b) \subseteq A \times B$ by

$$(X,b) = \{(x,b) \mid x \in X\}$$

and similarly for $X \subseteq A, Y \subseteq B$ we define (X,Y) to be $X \times Y$ and this also extends to higher order sets (sets of sets, sets of sets of sets, ...) in an inductive way. That is, if $\mathcal{A} \subseteq 2^A, \mathcal{B} \subseteq 2^B$, then

$$(\mathcal{A},\mathcal{B}) = \{(A,B) \mid A \in \mathcal{A}, B \in \mathcal{B}\} \text{ and } (\mathcal{A},B) = \{(A,b) \mid A \in \mathcal{A}, b \in B\}.$$

So, if $A = \{a,b\}$ then

$$(A,a) = \{(a,a),(b,a)\} \text{ and } (A,A) = \{(a,a),(a,b),(b,a),(b,b)\}.$$

The sort of inductive definition used above is also present in 2.1.4

Given two collections of subsets of a set, $\pi, \rho \subseteq 2^X$, then we define their product as $\pi.\rho \triangleq \{U \cap V \mid U \in \pi, V \in \rho\} - \emptyset$, the collection of all their non-empty intersections.

Another convention is that we use $\mathbb{R}^+$ to denote the set of positive real numbers.

Inducing

If we have some sort of structure, be it a partition, function, set, relation, etc., that "naturally" generates another structure then we call this process *induction.* We say that the resulting structure is induced by the original structure. As a convention we allow that any structure will induce itself. It should be clear from context whether induction will refer to this process, or the method of proof. This is used most often in 2.1.5.

If we have a definition for an "object" which consists of a set having certain properties, and we are given a particular object, A, then a "sub-object" of A is defined as a subset of A which also satisfies the properties of an "object". "Object" here can be replaced by an arbitrary structure.

2.1.2 Logical Connectives

Throughout the thesis we use $\vee$ for disjunction (Boolean OR), $\wedge$ for conjunction (AND), $\neg$ for negation (NOT), and $\Rightarrow$ for logical implication. We also use $\exists^{\omega}$ for "there exists infinitely many", $\exists^{<\omega}$ for "there exists finitely many", and $\exists!$ for "there exists a unique". If we have a collection of logical statements generated inductively from a collection of fundamental (or basic) statements, then the fundamental logical statements are called *atomic propositions.*

In other words, sentences in the logic will be generated by

$$B ::= p \mid B_1 \wedge B_2 \mid B_1 \vee B_2 \mid \neg B_1 \mid B_1 \Rightarrow B_2 \mid \exists x, B \mid \exists^{\omega} x, B \mid \exists^{<\omega} x, B \mid \forall x, B$$

where p stands for an atomic proposition, and x for a variable. For those unfamiliar with this notation, the logic above corresponds to the set of all statements that can be written using a finite number of atomic propositions combined using the operators discussed in the previous paragraph.

If we have a logical statement, δ, with the atomic propositions all being comparisons with members of a set, X, and a (total) map, $\eta : X \to Y$, from this alphabet to another, then we define $\eta[\delta]$ to be the logical statement formed by replacing every letter in δ with its image under η. For example, if $X = \{a, b, c\}$, $\delta = ((a < 5) \wedge (ab > 5))$ and $\eta : X \to X$ is given by $\eta(a) = \eta(b) = \eta(c) = c$, then $\eta[\delta] = ((c < 5) \wedge (c^2 > 5))$. If furthermore, $Y \subseteq \mathbb{R}$, either $\eta[\delta]$ is true, in which case we write $\eta \models \delta$ and say that η *models* δ; or $\eta[\delta]$ is false, in which case we write $\eta \not\models \delta$. A logical statement, δ is called *satisfiable* if there some η such that $\eta \models \delta$. We let $TRUE$, or T, be the logical statement corresponding to $\delta \vee \neg\delta$ (for some arbitrary δ), and this is modelled by every function. Dually, $FALSE$, or F, corresponds to $\neg T$, and no function models it.

As an example, if x, δ, η are as above; and $\alpha : X \to \mathbb{R}$ is given by $\alpha(a) = \alpha(c) = 1, \alpha(b) = 6$, then $\alpha \models \delta$, but $\alpha \not\models \eta[\delta]$.

We have the following proposition.

Proposition 2.1

Let δ be a logical statement such that $\alpha \models \delta$. The following hold :-

1. *If $\delta \Rightarrow \delta^*$ then $\alpha \models \delta^*$.*
2. *If $\alpha = \beta|_X$ for some set X, then $\beta \models \delta$.*

If δ is a logical statement over Σ, then we write Σ^δ to denote the set of all functions from Σ to $\mathbb{R}$ that model δ, so that $\Sigma^\delta = \{\eta \in \mathbb{R}^\Sigma \mid \eta \models \delta\}$. The intuitions behind the Boolean operators (eg. $B_1 \wedge B_2$ is true when B_1 is true or B_2 is true) generate relationships between sets of functions satisfying non-atomic propositions, as below.

Proposition 2.2

If δ and δ' are logical statements over Σ, then :-

1. $\Sigma^{\delta \vee \delta'} = \Sigma^\delta \cup \Sigma^{\delta'}$
2. $\Sigma^{\delta \wedge \delta'} = \Sigma^\delta \cap \Sigma^{\delta'}$
3. $\Sigma^{\neg\delta} = \mathbb{R}^\Sigma - \Sigma^\delta$
4. $\delta \Rightarrow \delta'$ *iff* $\Sigma^\delta \subseteq \Sigma^{\delta'}$

If $A \subseteq \Sigma$ and δ is a logical statement over Σ then we write $\Sigma^\delta|_A$ to denote the set of all functions from A to $\mathbb{R}$ that could be extended to satisfy δ. That is $\Sigma^\delta|_A = \{\eta \in \mathbb{R}^A \mid \exists \eta' \in \Sigma^\delta, \eta'|_A = \eta\}$. Thus one convenient way of thinking of $\Sigma^\delta|_A$ is as the projection of Σ^δ onto A. This is consistent with the induced notion of a set of restricted functions as introduced in 2.1.4.

This immediately gives us the following proposition.

Proposition 2.3

If δ is a logical statement over Σ and $A \subseteq \Sigma$ then $\Sigma^\delta \subseteq (\Sigma^\delta|_A, \mathbb{R}^{\Sigma - A})$.

If $A \subseteq \Sigma$ we write $\delta|_A$ for the logical statement formed by replacing sub-expressions over $\Sigma - A$ by T, so long as those sub-expressions are not equal to F. We can do this by writing first in the form $\delta_1 \vee \delta_2 ... \delta_n$ where each δ_i is the $\wedge$ of comparisons, at most one per clock.

Proposition 2.4

For any alphabet Σ, any logical statement δ, and any $A \subseteq \Sigma$ we have the following :

1. $\delta|_A \Rightarrow \delta$
2. $\Sigma^{\delta|_A} = (\Sigma^{\delta}|_A, \mathbb{R}^{\Sigma - A})$
3. $\nu \in \Sigma^{\delta|_A}$ *iff* $\nu|_A \models \delta|_A$

2.1.3 Strings

Throughout the thesis we will be considering strings of symbols/letters coming from a finite alphabet, denoted by Σ (with appropriate sub/super-scripts). The motive for the alphabet is that it will consist of all the possible interactions an abstract machine could have with the outside world. Since we are dealing with acceptors only, this means that Σ will be the set of all possible inputs to one of our abstract machines. We often write a in place of $\{a\}$ where $a \in \Sigma$.

If the string, σ, is finite, we call it a *word*, and denote its length by $l(\sigma)$. If the string is infinite we call it a *sequence* (we are only going to consider countably infinite sequences here). We denote the empty string by ϵ. ω is used here to represent the first infinite ordinal, which is isomorphic to the set of natural numbers (that is, we can regard ω as the set $\{0, 1, 2, \ldots\}$).

For example, if $\Sigma = \{a, b, c\}$ then $abca, aab, cc$ and a are all words, whilst $ababab\ldots, ccc\ldots$ and $abcabb\ldots$ are sequences.

If u is a string of any kind, then we let $u(i)$ stand for the ith member of the string[1], starting the count at 0. We use the product notation to stand for concatenation (so uv, or $u.v$, stands for the string resulting from concatenating u and v, where u is a word, and v is any string). Also, if u is a word then u^n is u repeated n-fold (so, for example $u^2 = uu$), and as a limiting case $u^\omega = uuuuuu\ldots$. We let $u^* = \{u^n \mid n \in \omega\}$, the set of words made up of finite copies of u concatenated together.

So, if $u = abca$ and $v = ccc\ldots$ then $u(1) = a, u(2) = b, v(1) = c, v(56) = c$ and so on. Also $uv = abcaccc\ldots, u^2v = abcaabcaccc\ldots$ and $u^\omega = abcaabcaabca\ldots$. For any string, u, $\epsilon u = u = u\epsilon$.

We extend all of these operations naturally to sets of strings, so if U and V are collections of strings then UV, or $U.V$, is defined to be $\{uv \mid u \in U, v \in V\}$. Similarly $U^n = \{u_1u_2\ldots u_n \mid u_i \in U\ \forall i \leq n\}$, and $U^\omega = \{u_1u_2u_3\ldots \mid u_i \in U\ \forall i \in \omega\}$. Now $U^* = \bigcup_{n\in\omega} U^n$ is the set of all words constructed via

[1]This is compatible with the isomorphism between sequences and functions from the index set.

finite concatenations of strings in U. Also, if U is a collection of strings, and $a \in \Sigma$, then aU would be $\{au \mid u \in U\}$.

For example, if $U = \{ab, ba\}$ and $V = \{cd, cc\}$ then

$$\begin{aligned} UV &= \{abcd, abcc, bacd, bacc\} \\ U^2 &= \{abab, abba, baab, baba\} \\ cU &= \{cab, cba\} \end{aligned}$$

If u is a sequence we let $In(u)$ be the set of all the symbols that occur infinitely often in u, that is $In(u) = \{a | \exists^{\omega} i \in \omega, u(i) = a\}$.

2.1.4 Relations and Functions

If we have two collections, X and Y, then a relation between X and Y is any subset, R, of (X, Y). We often write xRy if $(x, y) \in R$. Every relation, R, has an inverse, denoted R^{-1}, defined by

$$R^{-1} \triangleq \{(y, x) \mid (x, y) \in R\}$$

If X is a set, then an *equivalence relation* is any relation $R \subseteq (X, X)$ satisfying the following:

1. xRx for all $x \in X$ (reflexivity),
2. If xRy and yRz then xRy (transitivity), and
3. If xRy then yRx (symmetry).

A relation satisfying the first two conditions of an equivalence relation in addition to antisymmetry (aRb and bRa implies that $a = b$) is called a *partial ordering*. The paradigm case of a partial ordering is the class of all sets with inclusion as the relation.

We introduce the following partial order on collections of sets.

Definition 2.5

If $\pi, \rho \subseteq 2^X$ for some set X, then we say $\pi \leq \rho$ if for every $U \in \pi$ there is a $V \in \rho$ so that $U \subseteq V$. If $\pi \leq \rho$ then we say that π is finer *than ρ, and that ρ is* coarser *than π.*

Proposition 2.6

$\leq$ as defined in Definition 2.5 determines a partial order.

If a relation, R, has xRy *and* $xRz \Rightarrow y = z$ then we say that R is a *partial function*, and often denote $R(x)$ to be the unique element of Y related to x. If for every $x \in X$ there is a $y \in Y$ so that $y = R(x)$ then we say that R is a *total function*.

Every relation, R, induces a (total) function from 2^X to 2^Y. To be consistent with the notation introduced in the next section, this function is defined so that, for each $U \in 2^X$,

$$R[U] = \{y \in Y \mid uRy \text{ for some } u \in U\}$$

Functions

Whenever we define a natural extension to a function, we use square brackets rather than round brackets for emphasis. Occasionally, we define a function using the $\mapsto$ notation

If $f : X \to Y$ then we say that f is *injective*, or 1-1, if for every $x, x' \in X$ if $f(x) = f(x')$ then $x = x'$. We say f is *surjective*, or onto, if for each $y \in Y$ there is some $x \in X$ so that $f(x) = y$. A *bijection* is an injective surjection.

If $f : A \to B$ is a function, then we extend f to map subsets of A to subsets of B, so that $f[X] = \{f(x) | x \in X\}$ for all $X \subseteq A$. We extend to collections of sets as well, so that if $\mathcal{F} \subseteq 2^A$, then $f[\mathcal{F}] = \{f[X] | X \in \mathcal{F}\}$ (and continue inductively to define $f[U]$ where U is a higher order set). We also extend f to map strings in A to strings in B, so that $f^* : A^* \to B^*$ (and equivalently $f^\omega : A^\omega \to B^\omega$) is defined by $\forall i < l(\alpha), (f^*[\alpha])^i = f(\alpha^i)$, and $l(f^*(\alpha)) = l(\alpha)$ (this last condition is not needed for f^ω).

For example let $f : \mathbb{R} \to \mathbb{R}$ be defined by $f(x) = x^2$. From the above example $f[[-1, 0.5)] = [0, 1]$ and if $\mathcal{F} = \{(0, 1], \{0\}, [0.5, 2)\}$ then $f[\mathcal{F}] = \{(0, 1], \{0\}, [0.25, 4)\}$

So if $f : \{a, b, c\} \to \{a, b, c\}$ is defined by $f(a) = b, f(b) = c$, and $f(c) = a$ then $f^*[aabcb] = bbcac$.

The induced functions of the previous section are consistent with those defined here, since for every function $f : X \to Y$ and each $U \subseteq X$

$$\{y \in Y \mid uRy \text{ for some } u \in U\} = f[U] = \{f(u) \mid u \in U\}$$

Because each function, $f : X \to Y$, is a relation it has an inverse, f^{-1}. Sometimes f^{-1} is a function in which case we say f is *invertible*. As discussed earlier, f^{-1} will induce a function from 2^Y to 2^X even if it is not itself a function.

Proposition 2.7

If $f : X \to Y$ then

1. *f^{-1} is a total function iff f is a bijection.*
2. *f^{-1} is a partial function iff f is an injection.*

If $f : B \to A$ and $A = (A_1, A_2, ..., A_n)$ then we define $P_i^f \in A_i^B$ so that $P_i^f(x) = (f(x))(i)$. That is, P_i^f is the function giving the ith component of f. Another way of characterising P_f^i is that it is $P_i \circ f$ where P_i is the ith projection operator ($P_i(x) = x(i)$), and $\circ$ denotes composition.

Thus if $f : \mathbb{R} \to \mathbb{R}^2$ is given by $f(x) = (x^2, 1 - x)$ then for each $x \in \mathbb{R}$, $P_1^f(x) = x^2$ and $P_2^f(x) = 1 - x$.

If f is a partial function from A to B (that is, f is a function from some subset of A to B), then we extend f to make it a total function, in the following way.

1. Add an additional element to every set, say $*$. This will denote an "undefined" or error symbol; or a sink state.
2. For each $x \in A \cup \{*\}$, define $f' : A \cup \{*\} \to B \cup \{*\}$ by $f'(x) = \begin{cases} f(x) & , x \in \text{Domain}(f) \\ * & , x \notin \text{Domain}(f) \end{cases}$

In fact, to avoid potential confusion we add the symbol $*$ to every set, and alter every function, be it partial or total, in the above way. So from now on, when we write $f : A \to B$ what we mean is, in the more standard notation $f' : A \cup \{*\} \to B \cup \{*\}$ where f' is defined as above. We reserve the notation $f \in B^A$ for total functions (that is, when $f^{-1}[\{*\}] = \{*\}$).

We use 0 to denote the zero function, and also have the following derived functions.

Definition 2.8

If $f \in \mathbb{R}^X$ and $k \in \mathbb{R}$, then we use the following notation:

1. *$f + k \in \mathbb{R}^X$ is defined by $(f + k)(x) = f(x) + k$. Similarly $f - k$ is defined as $f + (-k)$.*
2. *$kf \in \mathbb{R}^X$ is defined by $(kf)(x) = kf(x)$.*
3. *If $S \subseteq X$ then $[S \to 0]f \in \mathbb{R}^X$ is defined by $([S \to 0]f)(x) = \begin{cases} 0 & , x \in S \\ f(x) & , x \notin S \end{cases}$*
4. *If X is discrete and well-ordered (eg. any subset of ω), then $\Delta f \in \mathbb{R}^{X - \{sup(X)\}}$ is defined by $(\Delta f)(x) = f(x+1) - f(x)$ where $+1$ denotes the successor function, and sup the supremum function.*

We can regard any string as a function from the index set into the alphabet. For example, $\sigma = bac$ corresponds to the function $\sigma : \{0,1,2\} \to \{a,b,c\}$ defined by $\sigma(0) = b, \sigma(1) = a, \sigma(2) = c$. This is a natural isomorphism, and we feel it is intuitive enough that we use the same symbol to refer to both the string and the function.

Proposition 2.9

If σ is a string over Σ (regarded as a function), and $f \in X^\Sigma$, then we have the following.

1. *If σ is a word, $f^*(\sigma) = \sigma \circ f$*
2. *If σ is a sequence, $f^\omega(\sigma) = \sigma \circ f$*

Combining two functions.

There are a number of ways of combining two functions, some of which we describe below. Not all of them are necessary for the later results, but they are included here for the sake of completeness.

For each $(f_1, f_2) \in (Y^{X_1}, Y^{X_2})$ there is a unique $f \in Y^{X_1 \uplus X_2}$ so that for each $x \in X_1 \uplus X_2$,

$$f(x) = \begin{cases} f_1(x) & , x \in X_1 \\ f_2(x) & , x \in X_2 \end{cases}$$

This is a natural isomorphism, and is described in the following picture.

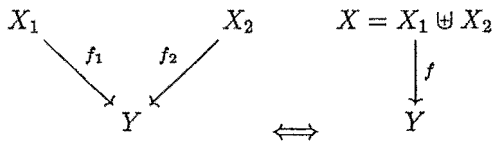

Figure 2.1: Functions over disjoint sets.

Given $(f_1, f_2) \in (Y^{X_1}, Y^{X_2})$ we denote the unique member of $Y^{X_1 \uplus X_2}$ described above as $f_1 \uplus f_2$.

Given $f \in Y^{X_1 \uplus X_2}$ we denote the unique pair of functions from Y^{X_1} and Y^{X_2} that f corresponds to as $f|_{X_1}$ and $f|_{X_2}$ respectively. Note that this is consistent with the typical notation for function restriction. As operators $|$ binds stronger than $\uplus$, giving us the following proposition.

Proposition 2.10

For any $f \in Y^{X_1 \uplus X_2}$ and $(f_1, f_2) \in (Y^{X_1}, Y^{X_2})$ we have the following.

1. $f|_{X_1} \uplus f|_{X_2} = f$.
2. $(f_1 \uplus f_2)|_{X_1} = f_1$ *and* $(f_1 \uplus f_2)|_{X_1} = f_2$

There is a dual process as well. For each $(f_1, f_2) \in (Y_1^X, Y_2^X)$ there is a unique $f \in (Y_1, Y_2)^X$ so that for each $x \in X$

$$f(x) = (f_1(x), f_2(x))$$

This is another natural isomorphism, described with the following picture.

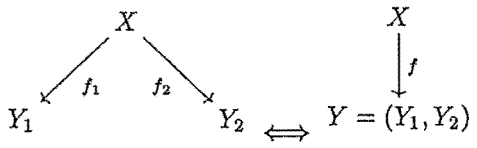

Figure 2.2: Functions from one set.

Given $(f_1, f_2) \in (Y_1^X, Y_2^X)$ we denote the unique member of $(Y_1, Y_2)^X$ described above as $f_1 \oplus f_2$.

Given $f \in (Y_1, Y_2)^X$ we denote the unique pair of functions from Y_1^X and Y_2^X described above as P_f^1 and P_f^2. This is consistent with our earlier use of P as a projection operator. Here, we let the projection operator bind stronger than $\oplus$, giving the following proposition.

Proposition 2.11

For any $f \in (Y_1, Y_2)^X$ and $(f_1, f_2) \in (Y_1^X, Y_2^X)$ we have the following.

1. $P_f^1 \oplus P_f^2 = f$.
2. $P_{f_1 \oplus f_2}^1 = f_1$ *and* $P_{f_1 \oplus f_2}^2 = f_2$.

There is one last way of combining two functions introduced here. If $f_1 \in Y_1^{X_1}$ and $f_2 \in Y_2^{X_2}$ then there is a unique $f \in (Y_1, Y_2)^{(X_1, X_2)}$ so that, for each $(x_1, x_2) \in (X_1, X_2)$,

$$f(x_1, x_2) = (f_1(x_1), f_2(x_2))$$

This isomorphism is described by the following picture.

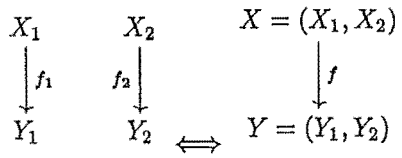

Figure 2.3: Functions over parallel sets.

Given f_1 and f_2 we describe the unique f described above as $f_1 \| f_2$.

Given f we describe the unique pair of functions f_1 and f_2 described above as $\diamond_1 f$ and $\diamond_2 f$ respectively. $\diamond$ binds stronger than $\|$, giving the following proposition.

Proposition 2.12 (F)

or any $(f_1, f_2) \in (Y_1^{X_1}, Y_2^{X_2})$ *and* $f \in (Y_1, Y_2)^{(X_1, X_2)}$ *we have the following.*

1. $\diamond_1 f \| \diamond_2 f = f$.
2. $\diamond_1(f_1 \| f_2) = f_1$ *and* $\diamond_2(f_1 \| f_2) = f_2$.

2.1.5 Partitions

If X is a set then a partition of X, π, is a collection of subsets that are pairwise disjoint and cover X. That is, if π is a partition, then $X = \bigcup \pi$ and for each $U, V \in \pi$, either $U = V$ or $U \cap V = \emptyset$. If $x \in X$, then we denote $\pi|_x$ to be the (unique) member of π that contains x. $A \subseteq X$ gives $\pi|_A = \{\pi|_a \mid a \in A\} \subseteq \pi$. This continues inductively for higher order sets. So, for example, if $\mathcal{A} \subseteq 2^X$ then $\pi|_{\mathcal{A}} = \{\pi|_A \mid A \in \mathcal{A}\}$. There are two *trivial* partitions of a set X, $\top_X \triangleq \{X\}$ and $\bot_X \triangleq \{\{x\} \mid x \in X\}$—every other partition is non-trivial.

Definition 2.13

If π *and* ρ *are two partitions of a set,* X*, then we define their* product *to be their product as members of* 2^{2^X} *as defined in 2.1.1. That is,*

$$\pi.\rho \triangleq \{U \cap V \mid U \in \pi, V \in \rho\} - \emptyset$$

Two collections of subsets of a set X *are called* orthogonal *if there product is* $\bot_X$.

Proposition 2.14

The product of two partitions of a set, X*, is a partition of* X.

For example, if $X = \{a, b, c, d, e\}$ then $\pi = \{\{a, b\}, \{c\}, \{d, e\}\}$ is a partition of X, but $\{\{a, b\}, \{c\}, \{d, e\}, \{b, d\}\}$ is not a partition of any set. Also, $\pi|_b = \{a, b\}, \pi|_c = \{c\}$, and if $A = \{a, b, d\}$ then $\pi|_A = \{\{a, b\}, \{d, e\}\}$. Also $\top_X = \{\{a, b, c, d, e\}\}$, and $\bot_X = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. π is non-trivial. Also, if $\pi_1 = \{\{a, b\}, \{c, d\}\}$ and $\pi_2 = \{\{a\}, \{b, c, d\}\}$ (both partitions of $\{a, b, c, d\}$), then $\pi_1.\pi_2 = \{\{a\}, \{b\}, \{c, d\}\}$, so that π_1 and π_2 are not orthogonal.

For the sake of convenience, we relax the formal definition slightly here and allow that two partitions, π_1 and π_2, are the same if $\pi_1 \Delta \pi_2 \subseteq \{\emptyset\}$. The formal definition requires that members of a partition be

non-empty, but this would force us to keep track of a number of cases in later discussions with partitions, and so we relax it slightly. We also identify any set containing $*$, the "undefined" symbol, with $*$, again for the sake of convienience.

Associated with any equivalence relation, $\equiv$, there is a natural partition $N(\equiv) = \{\{y \mid y \equiv x\} \mid x \in X\}$. The converse also holds, that is given a partition, π, there is a natural equivalence relation, $\equiv_\pi = \{(x,y) \mid \pi|_x = \pi|_y\}$.

For example, if $X = \{-2,-1,0,1,2,3\}$, and $a \equiv b$ if a and b have the same sign, then

$$N(\equiv) = \{\{-2,-1\},\{0\},\{1,2,3\}\}$$

As an example of the converse, if $\pi = \{\{a,b\},\{c,d\}\}$ as above, then

$$\equiv_\pi = \{(a,b),(b,a),(c,d),(d,c)\} \cup \{(x,x)|x \in \{a,b,c,d\}\} = (X,X) - (\{a,b\},\{c,d\})$$

Associated with any map, $\phi : X \to Y$, there is a natural partition of X, defined by

$$N(\phi) = \{\phi^{-1}[\{\phi(x)\}]|x \in X\}$$

So, if X is as above, and $f : X \to X$ is defined by $f(x) = |x|$ then $N(f) = \{\{0\},\{-1,1\},\{-2,2\},\{3\}\}$.

Here are some of the basic properties of partitions, which are to be used later on.

Proposition 2.15

If π_X and π_A are partitions of X and A respectively, then

1. *(π_X, A) (respectively (X, π_A)) is a partition of (X,A). This is the natural partition of (X,A) induced by π_X (respectively π_A).*
2. *(π_X, π_A) is a partition of (X,A). This is the natural partition induced by π_X and π_A.*

A Lattice of Partitions

A *lattice* is a collection of objects along with a partial ordering such that any pair of elements has both a least upper bound (lub) and a greatest lower bound (glb). The lattice is called *complete* if every set of objects has both a glb and a lub.

If we denote the lub and glb of x and y by $x \wedge y$ and $x \vee y$ respectively then the lattice is called *distributive* if for each x, y, z in the lattice $\wedge$ and $\vee$ satisfy the following two properties.

1. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$.

2. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

$\wedge$ and $\vee$ are binary operators. There extension to set operators are denoted $\bigwedge$ and $\bigvee$ respectively.

For any complete lattice, if $\wedge$ is defined then $\vee$ is automatically defined, so that $x \vee y = \bigwedge\{z \mid x \leq z, y \leq z\}$. Note that once glb's exist, this means that lub's exist so long as the lattice has a maximal element.

As the title of this section suggests - there is a lattice on the the set of partitions of a set, as below.

Proposition 2.16 (Shi87)

For any set X,

1. *2^{2^X} with order $\leq$ (as defined in Definition 2.5) form a complete distributive lattice with $\pi \wedge \rho = \pi.\rho$ for each $\pi, \rho \in 2^{2^X}$.*

2. *$\Pi(X)$ is a complete sublattice of 2^{2^X}.*

The minimal and maximal elements of 2^{2^X} are $\emptyset$ and 2^X respectively. Similarly the minimal and maximal elements of $\Pi(X)$ are $\bot_X$ and $\top_X$ respectively.

Partitions as Inducers and Induced Partitions

If $U \subseteq X$ then the finest partition of X containing $\{U\}$ is the partition of X induced by U, and is denoted Π_X^U.

Proposition 2.17

$\Pi_X^U = \top_U \vee \bot_X = \{U\} \cup \{\{x\} | x \in X - U\}$

If $\pi \in \Pi(X)$ and $f : X \to Y$ then the finest partition on Y, π', that makes $R \subseteq (\pi, \pi')$ defined by

$$R = \{(U, V) \mid f[U] \cap V \neq \emptyset\}$$

a function from π to π' is the partition of Y induced by π via f. One way of interpreting this is that each member of π must map under f into a unique member of π', and that f' maps the member of π to the corresponding member of π'.

We denote the partition π' induced as above by $\pi_{(f)}$.

Proposition 2.18

$\pi_{(f)}$ is characterised by $\bigwedge_{U \in \pi} \Pi_Y^{f[U]}$

If π_X and π_Y are partitions of X and Y respectively and $f : X \to Y$ then we say that π_X and π_Y are *well defining* with f if $f_{\pi_X,\pi_Y} = (\pi_X, \pi_Y)|_f$ is a function.

Proposition 2.19

π_X and π_Y are well defining with f iff $(\pi_X)_{(f)} \leq \pi_Y$.

If π induces partitions π_X of X and π_Y of Y then we say π is well defining with $f : X \to Y$ if π_X and π_Y are well defining with f. We write $f_{(\pi)} = (\pi_X, \pi_Y)|_f$.

One may note that if $\pi_Y = (\pi_X)_{(f)}$ then π will always be well defining with f, but where this definition will be used is when π induces π_Y via some other mechanism (other than with f).

Proposition 2.20

π is well defining with f iff for every $U \in \pi_X$ and $V \in \pi_Y$, if $f[U] \cap f[V] \neq \emptyset$, then $f[U] \subseteq f[V]$.

Now, if π and π' are partitions of X and Y respectively; and $f : X \to Y$; then because $f \subseteq (X, Y)$ then $f_{\pi,\pi'} = (\pi, \pi')|_f \subseteq (\pi, \pi')$. $f_{\pi,\pi'}$ is not typically a function, but when $f_{\pi,\pi'} : \pi \to \pi'$ we say that π and π' are well defining with f and together induce the function $f_{\pi,\pi'}$.

Proposition 2.21

π and π' are well defining with f iff π is well defining with f and $\pi_f \leq \pi'$.

Now, if the π and π' above are induced by a single partition $\pi^\star$ then we again say that $\pi^\star$ is well defining with f and induces the function $f_{\pi^\star} = f_{\pi,\pi'}$.

2.2 Machines

In the following sections we will define various classes of abstract machines (or automata), but there are a lot of concepts which overlap and so it is pertinent to discuss them generically here, leaving the specifics for the following sections.

We are restricting our attention to acceptors rather than transducers. *Acceptors* are machines that receive an input and then either accept or reject it. *Transducers* are a more general class of machines, which receive an input, manipulate it in some manner, and then return an output.

However, the full generality of transducers is not required to determine most important structural relationships as the difference is largely superficial, and would tend to obscure the results that we are attempting to illuminate here. The results we are attempting to extend here are presented for transducers in [Shi87].

We denote the set of all the inputs accepted by a machine as the *behaviour* or *language* of that machine (this is largely because inputs correspond to actions), and the set of all the inputs accepted by any one of a class of machines as the *expressiveness* of that class. If the machine is $\mathcal{M}$ then its language is denoted $L(\mathcal{M})$.

2.3 Finite Machines

In this section we review background material related to finite machines. All the technical details can be found in [HU79], albeit using slightly different notation.

A first glance at the nomenclature above would lead one to think that we are going to be considering infinite machines. Indeed we are, but only in the sense that infinite machines deal with sequences whereas finite machines deal with words (remember that words are finite, whereas sequences are infinite). Throughout, we only consider machines which can be specified by a finite tuple of finite sets. There is related work extending machines to allow for specifications containing infinite objects, for example Turing Machines and Push Down Automata [HU79, Kel95, Shi87], but they are not considered here.

Definition 2.22

A finite machine, $\mathcal{M}$, is a non empty 5-tuple (Q, Σ, E, s, F) where :

1. *Σ is a finite input alphabet.*
2. *Q is a finite set of states*
3. *$E \subseteq (Q, \Sigma, Q)$ is the set of transition edges.*
4. *$s \in Q$ is the initial state.*
5. *$F \subseteq Q$ is the set of accepting states.*

We write $q \xrightarrow{a} r$ if $(q, a, r) \in E$, and say that machine $\mathcal{M}$ in state q can *exhibit* an a and move to state r. This is called a *transition*. If w is a string of length n we write $q \stackrel{w}{\Longrightarrow} r$ if there exists $q(0)q(1)\ldots q(n)$ where $q(0) = q$, $q(n) = r$ and $q(i) \xrightarrow{w(i)} q(i+1)$ for all $0 \leq i < n$.

If $\alpha \in \Sigma^*$ then α is *accepted* by $\mathcal{M}$ if there is some $q \in F$ so that $s \stackrel{\alpha}{\Longrightarrow} q$. A language, $L \subseteq \Sigma^*$, is *regular* if there is some finite machine accepting it. These languages are called regular because they are

precisely those defined recursively by the regular expressions, as shown below [Kel95]:

$$A ::= p \mid A_1 \cup A_2 \mid A_1 A_2 \mid A^*$$

where p is one of $\emptyset$, $\{\epsilon\}$, or $\{a\}$ for some $a \in \Sigma$. The regular languages are closed under finite Boolean operations.

Proposition 2.23 (HU79)

If L_1 and L_2 are regular languages over Σ, then so are $L_1 \cup L_2, L_1 \cap L_2$ and $\Sigma^* - L_1$.

A machine, $\mathcal{M}$, is *deterministic* if E is a partial function from (Q, Σ) to Q. A machine that is not deterministic is called *non-deterministic*, but henceforth, we assume every machine is deterministic. If E is a partial function, we turn it into a function as described in 2.1.4 and use the letter f to denote the resulting function. So we have $f : (Q, \Sigma) \to Q$.

Henceforth, we will only use the letters $\mathcal{M}$, Q, Σ, f, s, and F for describing the components of a machine (be it a finite, timed (Definition 2.32), or ω-machine (Definition 2.24)) as given above. We will label machines by sub-scripting with labels, and we carry those labels through to the components of the machine and back again (so that $\mathcal{M}_1$ has components Q_1, Σ_1, f_1, s_1, and F_1, and F_* comes from machine $\mathcal{M}_*$).

We can describe a finite machine as an edge-labelled digraph, with Q the vertices, E the edges, with s and F marked. So, for example, if $\mathcal{M}$ is the finite machine defined by

$$Q = \{o, p, q, r\}, \Sigma = \{a, b, c\}, s = o, F = \{o\}, E = \{(o, a, p), (o, b, q), (p, b, r), (q, a, r), (r, c, o)\}$$

then we can represent $\mathcal{M}$ in the following manner.

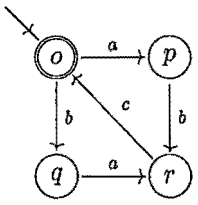

Figure 2.4: Example of a finite machine.

In either case $L(\mathcal{M}) = ((ab + ba)c)^*$. Note we observe the following conventions when describing $\mathcal{M}$ from a graph.

1. States are circled, except for the "undefined" state, which is not present.

2. Transitions are arrows, except for the "undefined" transitions which account for all the transactions not displayed.

3. The starting state is denoted by an inward arrow.

4. The final states have two rings around them.

Here is a second example.

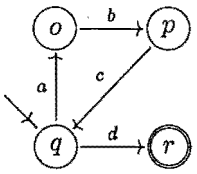

Figure 2.5: Example of a finite machine.

The finite machine corresponding to Figure 2.5, $\mathcal{M}$, is represented by $\Sigma = \{a, b, c, d\}$, $Q = \{o, p, q, r\}$, $E = \{(o, b, p), (p, c, q), (q, a, o), (q, d, r)\}$, $s = q$, and $F = \{r\}$. The language of $\mathcal{M}$ is $(abc)^*d$.

2.4 ω Machines

We present a quick review of ω-machines. More technical details can be found in [Tho90]. We define ω-machines as variants of finite machines that work with sequences instead of words. As a result, we no longer have final states. There are assorted adjustments we may make. The first of these keeps the set of accepting states, but now a word is accepted when one of these states is reached infinitely often.

Definition 2.24

A Büchi machine, $\mathcal{M}$, is a 5-tuple (Q, Σ, f, s, F) where Σ, Q, f, s, and F are as in Definition 2.22.

We define *deterministic/non-deterministic* Büchi machines based on whether f is a partial function or not—analogously with how we defined their finite counterparts. We can add an absorbing state to make f a function as before, and henceforth assume that $f : (Q, \Sigma) \to Q$. We also use the same notation for transitions.

The differences between finite and Büchi machines arise because Büchi machines accept languages from Σ^ω rather than Σ^*. Thus, iff $\alpha \in \Sigma^\omega$, then $\sigma \in Q^\omega$ is a *run* of M on α if

$$\sigma(0) = s \text{ and } \forall i \geq 0,\ \sigma(i) \xrightarrow{\alpha(i)} \sigma(i+1)$$

Note that this is the same as for Finite Machines excepting the range of the index. A run is called a *successful* run if $In(\sigma) \cap F \neq \emptyset$ (that is, at least one state in F is reached infinitely often by σ), and M *accepts* α if there is a successful run of M on α. We say the language *recognized* by M is $L(M) = \{\alpha \in \Sigma^\omega \mid M \text{ accepts } \alpha\}$. A language is *Büchi recognizable* if there is some Büchi machine recognising it.

For example, if we regard Figure 2.4 as a Büchi machine, then the language accepted by $\mathcal{M}$ will be $(\{ab, ba\}c)^\omega$ rather than $(\{ab, ba\}c)^*$ as there is no finite behaviour for an ω-machine, and no infinite behaviour for a finite machine. Similarly, the language accepted by the Büchi machine described in Figure 2.5 is empty.

The set of Büchi recognizable languages is closed under finite intersections, unions, and complements. It is also linked closely to the set of languages recognized by finite machines, as shown in the following proposition.

Proposition 2.25 (Tho90)

1. *If $V \subseteq A^*$ is regular, then V^ω is Büchi recognizable.*
2. *If $U \subseteq A^*$ is regular, and $L \subseteq \Sigma^\omega$ is Büchi recognizable, then $U.L$ is Büchi recognizable.*
3. *If $L_1, L_2 \subseteq A^*$ are Büchi recognizable, then so are $L_1 \cup L_2$ and $L_1 \cap L_2$.*
4. *If $L \subseteq \Sigma^\omega$ is Büchi recognizable, then so is $\Sigma^\omega - L$.*
5. *A language is Büchi recognizable iff it is of the form $\bigcup_{i=1}^{n} U^i.(V^i)^\omega$ where U^i and V^i are regular, and $V^i.V^i \subseteq V^i$.*

All of these are proved by constructions, the details being present in [Tho90]. A problem arises because the set of languages accepted by (deterministic) Büchi machines is not closed under complementation.

The languages accepted by non-deterministic Büchi machines are often referred to as the regular ω-languages. This definition is reinforced by the closure properties they share in common with the regular languages (Proposition 2.25).

From Proposition 2.25, we see that any non-empty regular ω-language contains an ultimately periodic sequence (that is, of the form $uvvv\ldots$ where for some i, $u \in U^i, v \in V^i$), and thus the emptiness problem for them is decidable. Proposition 2.25 also implies that the languages recognized by Büchi machines are closed under finite Boolean operations, and so the inclusion/equivalence problems for them are decidable.

By varying the acceptance criteria, we develop alternative ω-machines. For a Büchi machine we have a set of accepting states, at least one of which must be reached infinitely often for a sequence to be

accepted. If we change this to a collection of accepting *sets* of states, one of which is precisely the set of states reached infinitely many times, then the determinism-complementation problem disappears. This new criteria is called Muller acceptance criteria and the resulting machines Muller machines.

Definition 2.26

A Muller machine *is a tuple of the form* $M = (Q, \Sigma, s, f, \mathcal{F})$ *where*

1. Q, Σ, s, *and* f *are as in Definition 2.22.*
2. $\mathcal{F} \subseteq 2^Q$ *is the collection of all accepting sets of states.*

Runs on Muller machines are the same as runs on Büchi machines, but now a run, σ, is *successful* if $In(\sigma) \in \mathcal{F}$. Apart from this *Muller acceptance* and *recognizability* are defined analogously with their Büchi counterparts. Now, any language accepted by a Muller machine is a language accepted by a deterministic Muller machine (to show this take the power set of the states of the non-deterministic machine as the states of your deterministic machine), and so we do not need to consider non-deterministic Muller machines (determinism makes the theory, as well as the computation, a lot easier). Also, the following theorem (McNaughton's Theorem) shows us that we lose nothing by looking at Muller machines rather than Büchi machines.

Theorem 2.27 (Tho90)

An ω*-language is Muller recognizable iff it is regular.*

For example if we regard the machine in Figure 2.4 as a Büchi machine then it accepts the same language as the Muller machine, $\mathcal{M}$, where $Q = \{o, p, q, r\}, \Sigma = \{a, b, c\}, s = o, \mathcal{F} = \{U \mid U \in 2^Q, o \in U\}$, and $E = \{(o, a, p), (o, b, q), (p, b, r), (q, a, r), (r, c, o)\}$. Notice that the only difference is in the $\mathcal{F}$ rather than F. If we kept the same specifications, apart from $\mathcal{F}$ which we change to be $\{\{o\}\}$ then $L(\mathcal{M})$ would now be empty.

If $\mathcal{F}$ were changed to be $\{\{o, p, r\}\}$ then $L(\mathcal{M})$ would be $((ab \cup ba)c)^*(abc)^\omega$. We can construct a Büchi machine with the same behaviour as this new machine, for example Figure 2.6, but no deterministic Büchi machine will accept the same language (left as an exercise for disbelievers). Intuitively this arises because after any amount of processing of $(abc)^\omega$ the machine must still be able to process both the rest of $(abc)^\omega$ and also $(abc)^n bac(abc)^\omega$ for a sufficiently large n.

To illustrate the fact that the expressiveness of deterministic Muller machines is as great as that of their non-deterministic counterparts we give the following example. If $\mathcal{M}$ is the non-deterministic Muller machine represented in Figure 2.7 with $\mathcal{F} = \{\{1, 3\}, \{2\}\}$ then $L(\mathcal{M}) = ((ac)^*a(\epsilon \cup b)a^*b)^*(ac)^\omega \bigcup (ac)^*a(\epsilon \cup b)(b(ac)^*a(\epsilon \cup b))^*a^\omega$. This, however, is precisely the language accepted by the deterministic machine $\mathcal{M}_*$,

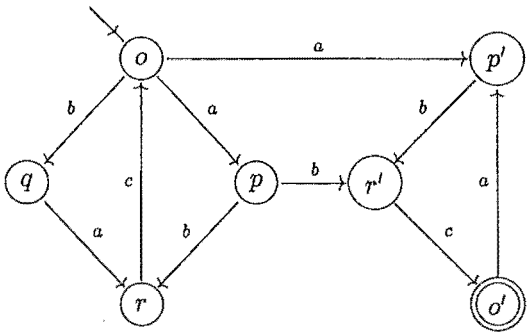

Figure 2.6: Example of a non-deterministic Büchi machine equivalent to the Muller machine in Figure 2.4 with $\mathcal{F} = \{\{o, p, r\}\}$.

represented in Figure 2.8 with $\mathcal{F}_* = \{\{1, 23\}, \{2, 123\}\}$. Note how describing a machine purely by it's language quickly becomes very cumbersome for any realistic or even slightly more complicated machine.

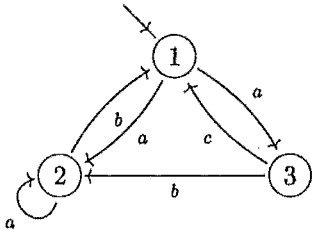

Figure 2.7: A non-deterministic Muller machine

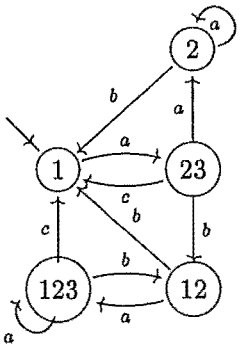

Figure 2.8: A deterministic Muller machine with the same language as Figure 2.7

There are other specifications equivalent (in terms of languages recognized by) to Büchi and Muller machines—for example, Rabin machines [Tho90]. For the remainder of this work we use deterministic Muller machines, as they seem the more natural to work with.

2.5 Timed Machines

Although the above descriptions are very useful, there is a natural desire to make ones' models as realistic and natural as possible. The behaviour of many systems depend on time, and this motivates attempts to introduce time into the abstract specifications of machines. There are numerous ways we can attempt to do this. We could discretize the times involved in any situation we are attempting to model, and describe the system as a very large finite or ω-machine. In many situations, however, this is not the most natural approach (quite apart from not providing an excuse for further exploration). In this thesis we will be considering a specification for timed machines that allows for arbitrary (real) times to be considered. The model was initially proposed in [AD94], and a summary of pertinent parts of this paper follows, accompanied by running examples.

Rather than modelling merely a sequence of actions, we now wish to model a sequence of $(action, time\ of\ occurence)$ pairs. Instead of restricting the possible occurrence times to be integral (and thus discrete), we allow the times to range over the reals. We will, however, impose a couple of weak conditions on time sequences. They are properties standard time sequences satisfy, so this should not present a problem in most modelling situations. Note that implicit in the previous discussion (and the following lemma) is the assumption that we are modelling systems with an infinite behaviour. That is, their specifications as abstract models do not halt.

Definition 2.28

A time sequence is a sequence of non-negative real numbers, τ, satisfying :

1. *Monotonicity:* $\forall i, \tau(i) < \tau(i+1)$. *That is, time always increases.*

2. *Progress:* $\forall t \in \mathbb{R}, \exists i \in \omega, t < \tau(i)$. *The sequence of times increases without bound.*

The progress constraint prevents Zeno computations. That is, if this condition were not imposed the time sequence might have a limit, forcing infinite actions to occur in a finite time, which would cause problems when considering time past this limit (Zeno's paradox).

The monotonicity constraint on the times can be replaced by requiring only that the time sequence be non-decreasing rather than strictly increasing, and there is no particular reason for choosing this specification over the other (apart from the necessity of choosing one, and the intuitive feeling that any action must take some time). All the results carry through to this alternative specification with little change. We use the strictly increasing specification given in Definition 2.28.

Definition 2.29

A timed word, (α, τ), *is a member of* $(\Sigma^\omega, (\mathbb{R}^+)^\omega)$ *where* Σ *is finite, and* τ *is a time sequence. A set of timed words is a* timed language.

In order to clarify any points of uncertainty arising from these definitions, we illustrate with an example. Suppose we are attempting to model the situation depicted in Figure 2.9.

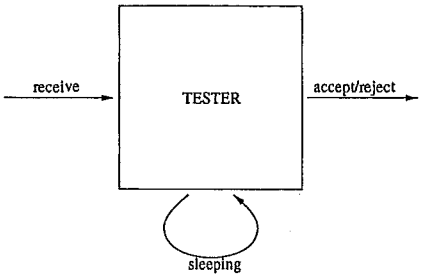

Figure 2.9: Simple Testing Station

The Tester in Figure 2.9 stands at the end of a production line, and receives products from the line (the "*receive*" arrow). It then tests to see if the product is satisfactory or not. If it is, then it "*accepts*" the product—if not, it is "*reject*"ed. It always takes less than 1 time unit to test an object, and if the tester is left idle for 4 time units then it goes into a sleeping state. A valid timed word representing a possible behaviour of the TESTER is, for example

$$(receive, 20), (accept, 20.1), (sleep, 30), (receive, 50), (accept, 50.1), (sleep, 60), ...$$

where the cycle of sleeping, receiving and acceptance continues on endlessly. Note that the sequence requirements on the times means that acceptable words are infinite, so that a simple $(sleep, 10)$ move, which intuitively we might suspect to correspond to the TESTER going to sleep after 10 time units, and never awakening, is *not* a valid timed word. Thus some care must be taken in applying these specifications to modelling situations. An example of an unacceptable timed word might be

$$(receive, 20), (accept, 22), sleep(30), (receive, 40), (accept, 42), sleep(50), ...$$

because the description of the system specified that the testing take no more than 1 time unit. It is, however, still a valid timed word. Examples of invalid timed words are

$$(receive, 20), (accept, 20.1), sleep(30), (receive, 20), (accept, 20.1), sleep(30), ...$$

because the sequence of times is not an increasing one; and

$$(receive, 20), (accept, 22), sleep(22.2), (receive, 22.22), (accept, 22.222), sleep(22.2222), ...$$

because the sequence of times is bounded.

In order to develop abstract machines that work with timed sequences as described above, we need to adapt the specifications for such machines. The states are suitable as they are. For example, TESTER above might have 3 states—an idle state, a sleeping state, and a test state. We, however, need to include something to account for the time sequence associated with transitions between these states.

We do this by including a set of stop-watches, called clocks in [AD94], the current value of which can be both checked and reset during transitions. We formalise this as follows.

Definition 2.30

If we are given a set of stopwatches, *C, then a* time state, *ν, is a map from C to* $\mathbb{R}$. *For each $x \in C$, $\nu(x)$ will be the reading of the stopwatch x in the state ν.*

Although we prefer the term stopwatch, we use clock synonymously to be consistent with [AD94].

In the TESTER example above, we will be using two clocks—one for ensuring that the testing takes less than 1 time unit, and another for ensuring that the tester sleeps if left idle for more than 4 time units. Actually, the description of TESTER only requires a single clock (which can be used to ensure both conditions above), but we use two anyway. The dual motives being style (so that there is a separate clock for each part, a more intuitive way of building the model), and illustration. If the 2 clocks used were called x and y (in that order), then a valid time state, ν, would be any map from $\{x, y\}$ into $\mathbb{R}$, with $\nu(x)$ being the time as measured by x, and $\nu(y)$ the time as measured by y.

Note that $[X \to 0]\nu$ as defined in Definition 2.8 will correspond to resetting the stopwatches in X and leaving the other stopwatches untouched.

The change in specifications will mean that transitions will now cause a change from one (state,time state) pair to another, rather than just a change from one state to another as was the case for finite and ω-machines. Thus we describe a (state,time state) pair as an *extended state.*

Thus, for the TESTER example above, $(idle, (0, 0))$ would be an extended state, as would $(test, (30, 5.4))$.

Now, to incorporate the stopwatches into the model so that they can enable, or disable, certain transitions (as opposed to merely recording times), we attach constraints to each edge, and for a particular edge to be traversed (that is, for a transition to take place) we will require that the appropriate action is the

next input, *and* also that the time state will satisfy the constraint corresponding to that edge. In order to be able to model delays we also allow any stopwatches to be reset as an edge is traversed. Convention (and utilisation of all present information) dictates that the stopwatches are reset immediately after the constraint is checked. The allowable stopwatch constraints are built up as follows.

Definition 2.31

If X is a set of stopwatch variables, the set of stopwatch constraints, $\Phi(X)$, is built up inductively from the following,

$$L ::= p \mid \neg L \mid L_1 \vee L_2$$

where p is $(x < c)$ or $(x > c)$ for some stopwatch variable x and some rational constant c.

In a logical statement, strings of the form of the first two items above are the atomic propositions.

We restrict comparisons to be over the rationals to enable some decision procedures. Extending the stopwatch constraints to include formulae of the form $(x \leq y + c)$ where x and y are stopwatches and c is a rational constant, does not increase the expressiveness of the machines considered, so we make do without allowing them. Allowing constraints of the form $(x + y \leq c)$ increases the expressiveness, but many decidability problems arise, and so we do not consider them. These issues are discussed in detail in [AD94].

Just as with the ω-machines of Section 2.4 we can define a number of different acceptance conditions (we are again accepting sequences rather than words)—Büchi , Muller, and Rabin [Tho90] conditions amongst others. Here we give only the specification for a timed Muller machine, and this specification will be the one we use for timed machines throughout.

Definition 2.32

A timed Muller machine *is a tuple of the form $M = (\Sigma, Q, s, C, E, \mathcal{F})$ where:*

1. *Σ, Q, s are as in Definition 2.22,*
2. *C is the finite set of* stopwatches,
3. *$E \subseteq (Q, Q, \Sigma, 2^C, \Phi(C))$ is the finite set of* edges,
4. *$\mathcal{F} \subseteq 2^Q$ is as in, Definition 2.26, and so gives the set of all possible combinations of states reached infinitely often.*

The intuition behind an edge, (q, q', a, X, δ) is that the machine is able to move from state q to state q' upon exhibiting(receiving) an a, so long as the constraint δ is satisfied. In doing so, the stopwatches in

X will be reset. Thus, for extended states (q,η) and (r,ν) we write $(q,\eta) \xrightarrow{a,t} (r,\nu)$ if there is an edge $e = (q,r,a,X,\delta)$ where $\eta + \Delta\tau \models \delta$ and $\nu = [X \to 0](\eta + t)$. This will mean the machine may move from extended state (q,η) to extended state (r,ν) upon receipt of an a and waiting for t time units.

Now, for the TESTER example, we can describe it abstractly as the timed Muller machine, $\mathcal{M}$, where

1. $\Sigma = \{receive, accept, reject, sleep\}$. To shorten the later descriptions we often abbreviate *receive* as b, *accept* as $\surd$, *reject* as $\times$, and *sleep* as ZZZ.
2. $Q = \{sleeping, test, idle\}$. Again, as abbreviations, we use A in place of *sleeping*, B in place of *test*, and C in place of *idle*.
3. $s = sleep$
4. $C = \{y, z\}$
5. $E = \{e_1, e_2, e_3, e_4, e_5\}$ where

$$\begin{aligned} e_1 &= (sleep, testing, receive, \{y\}, T) \\ e_2 &= (testing, idle, accept, \{z\}, y < 1) \\ e_3 &= (testing, idle, reject, \{z\}, y < 1) \\ e_4 &= (idle, sleeping, sleep, \emptyset, z \geq 4) \\ e_5 &= (idle, testing, receive, \{y\}, z < 4) \end{aligned}$$

6. $\mathcal{F} = 2^Q$. That is, all cycles will be accepted.

For TESTER we have

$$(sleep, 0) \xrightarrow{receive,5} (testing, (0,5))$$

(remember that 0 also denotes the zero function) and

$$(idle, (15,1)) \xrightarrow{sleep,4} (sleep, (19,5))$$

but it is not the case that

$$(idle, (15,0)) \xrightarrow{sleep,4} (sleep, (19,4))$$

or

$$(idle, (15,0)) \xrightarrow{sleep,4} (sleep, (19,5))$$

A timed machine is *deterministic* if for each extended state, and for each action, there only one potentially traversable edge. That is, for each $(q, \eta, a) \in (Q, \mathbb{R}^C, \Sigma)$ there is a unique $e \in E$ so that $q = P^1(e), a = P^3(e)$, and $\eta \models P^5(e)$. We only want to analyse deterministic machines here, and we define the transition function $f : (Q, \mathbb{R}^C, \Sigma) \to (Q, 2^C, \Phi(C))$ by $f(q, \eta, a) = (P^2 \uplus P^4 \uplus P^5)(e)$ where e is the unique edge discussed above. TESTER, as described above by timed machine $\mathcal{M}$, is deterministic. The transition function for $\mathcal{M}$ is defined by

$$f(q,\eta,a) = \begin{cases} (testing, \{y\}, T) & , \quad q = sleeping, a = receive \\ (idle, \{z\}, y < 1) & , \quad q = testing, \eta(y) < 1, a = \surd \\ (idle, \{z\}, y < 1) & , \quad q = testing, \eta(y) < 1, a = \times \\ (sleeping, \emptyset, z \geq 4) & , \quad q = idle, \eta(z) \geq 4, a = sleep \\ (testing, \{y\}, z < 4) & , \quad q = idle, \eta(z) < 4, a = receive \\ & , \quad otherwise \end{cases}$$

Note that f is a well-defined function, and that this is always the case if the machine in question is deterministic.

A *run* of (α, τ) on $\mathcal{M}$ is a sequence, σ, of extended states where $\sigma(0) = (s, 0)$ (note the zero function denoted by 0) and $\sigma(i) \xrightarrow{\alpha(i),(\Delta\tau)(i)} \sigma(i+1)$ for each $i \geq 0$. This is a successful run if $In(P^1_\sigma) \in \mathcal{F}$, and $\mathcal{M}$ accepts (α, τ) if there is a successful run of (α, τ) on $\mathcal{M}$.

Again referring back to the TESTER example, for the timed word

$$(receive, 20), (accept, 20.1), (sleep, 30), (receive, 50), (accept, 50.1), (sleep, 60), ...$$

(so that $\alpha = (receive, accept, sleep, ...)$, and $\tau = (20, 20.1, 30, 50, ...)$) , an example of a run of it on $\mathcal{M}$ is σ, where σ is defined by

$$\begin{aligned} \sigma(0) &= (sleeping, 0) \\ \sigma(1) &= (testing, (0, 20)) \\ \sigma(3i-1) &= (idle, (0.1, 0)) \\ \sigma(3i) &= (sleeping, (10, 9.9)) \\ \sigma(3i+1) &= (testing, (0, 29.9)) \end{aligned}$$

where i ranges over all positive integers. This is a successful run, since

$$In(P^1_\sigma) = In((sleeping, testing, idle, sleeping, testing, idle, ...)) = \{sleeping, testing, idle\} \in \mathcal{F}$$

Note that if a machine is deterministic then there precisely one run of any given word on that machine

(each successive extended state is determined by the previous one, the current action, and time state).

As with the abstract machines presented earlier, we can define a machine as a edge-labelled digraph, accompanied by the collection of accepting sets.

Continuing with the TESTER example, we could have described it using the following picture.

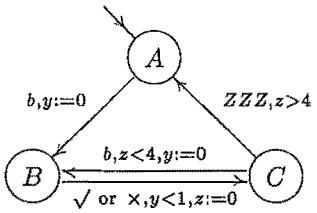

Figure 2.10: TESTER described as a digraph, all cycles accepted

Note the observations of the following conventions when using the digraph to define a timed machine.

1. States and the starting state are as for Finite Machines, Definition 2.22
2. The first and second components of an edge are the starting and ending states of that edge, and the other components are the labels attached to the edge.
3. If there is no constraint labelling an edge then the constraint is T

In a similar manner to the ω-machine case, the timed regular languages are defined to be those accepted by timed Büchi machines, and once again, equivalence between the Büchi acceptance conditions and the Muller acceptance conditions means that a language is regular iff it is accepted by a timed Muller machine. The timed regular languages are also closed under finite unions and intersections (proof by construction, given in [AD94], parallels closely the equivalent proof for Büchi machines). The timed regular languages, unlike their untimed counterparts, are not closed under complementation. The set of languages accepted by deterministic Muller machines, is closed under all Boolean operations, which is one of the reasons we choose this specification. A complete discussion of this is in [AD94].

Restricting the stopwatch constraints to range over the rationals only means that when checking for emptiness one only needs to check over rational times, and this is decidable (in fact one can assume that the stopwatch constraints are integral [AD94]).

2.5.1 The Region Machine

This section discusses how we can treat a timed machine as a special sort of finite machine. This does not mean in any way that we should abandon the specifications of timed machines, as both computationally

and structurally we lose a lot in the conversion. This machine is useful in examining the emptiness of the language of a timed machine.

Proposition 2.33 (AD94)

Let L be a timed regular language. For every word, $(\sigma, \tau) \in L$, there exists a time sequence $\tau' \in Q^\omega$ such that $(\sigma, \tau') \in L$.

This comes about because the finite number of edges imply that each term in the time sequence must lie either between two distinct rationals, or is equal to a rational. The rationals being dense suffices. See [AD94] for a formal proof.

If we let $t.\mathcal{M}$ be the machine formed by any constants in edge constraints by t, then we have the following result.

Proposition 2.34 (AD94)

For each machine, $\mathcal{M}$, and each positive rational t, $(\sigma, \tau) \in L(\mathcal{M})$ iff $(\sigma, t.\tau) \in L(t.\mathcal{M})$

Thus for a particular $\mathcal{M}$ we can pick a t so that each constraint only contains integer constants, and still have analogous behaviour. For the remainder of this section we assume that this has been done, and so we assume that constraint constants are integral.

Now, because the constraints only contain integer constants, there is only a certain granularity within which the the machine will either accept or reject all words. For a particular stop watch, x, we let $M_x = \max\{c \mid c \text{ is a stopwatch constant compared to } x$ and for time state ν let $C(\nu) = \{x \mid x \in C, x < M_x\}$. Then, we construct equivalence classes of extended states from the following equivalence relation.

$$(q, \nu) \equiv (q', \nu') \text{ iff } \begin{cases} q = q' \\ f(\nu, C(\nu)) = f(\nu', C(\nu')) \\ \nu(x) \mapsto \nu'(x) \textit{ is order preserving on } C(\nu) \cap C(\nu') \end{cases}$$

where $f(\nu) = \lfloor \nu(x) \rfloor$

There are a finite number of these equivalence classes, and states within the same equivalence class are treated the same by $\mathcal{M}$. That is, the strings accepted from any two equivalent states are identical.

Another way of describing this is that the granularity of $\mathcal{M}$ is at least that of the partition induced by $\equiv$. Using these equivalence classes as states one can construct the *region* machine (see [AD94] for details) which accepts α iff there is some τ so that (α, τ) is accepted by $\mathcal{M}$. Here is an example to illustrate this construction.

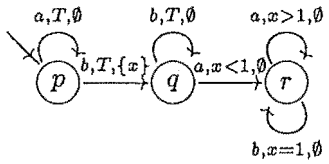

Figure 2.11: An example of a timed machine.

For the machine above let $\mathcal{F} = \{r\}$. Now there is only the one clock, x, and the largest value it is compared to is 1, so that $M_x = 1$. This region machine this generates is partially shown in Figure 2.12 (some transitions are excluded so that the diagram is easier to read). Note that some states are unreachable—in Figure 2.12, the state "r, $x = 0$" is unreachable.

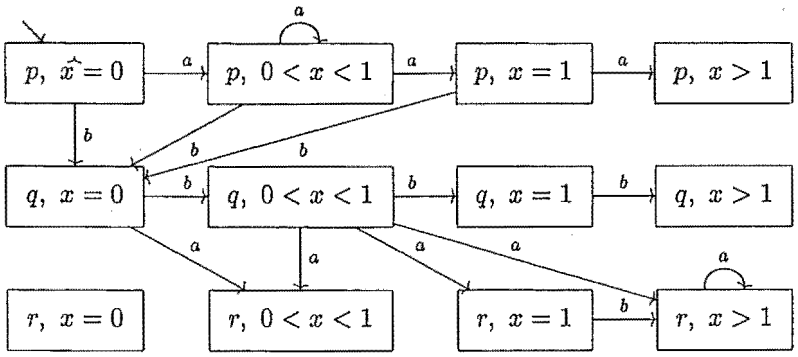

Figure 2.12: A partial region machine.

As you can see, the size of the region automata can be quite large. In fact, it is bounded by (and grows with order) $|Q||C|!2^{|C|}\Pi_{x \in C}(c_x + 2)$, shown in [AD94]. This limits the practical use of region machines, and was one of the biggest problems with the use of Finite Machines in the first place.

2.6 Summary

In this chapter we have described most of the mathematical concepts that we will be using for the remainder of the thesis. They are presented in mathematical context, and so not all of the material shown here will be used in the later chapters.

Many set theoretic concepts are fundamental to the theory we are developing—indeed, machines are presented here as collections of appropriate sets with attached meaning—and so we begin our chapter with a brief discussion of some of these. The notation used has been chosen to minimise the presentation (and hopefully the conceptual difficulties) of our later results, but the majority of this comes from [Hal60].

The logic used throughout is next introduced. Although this is only of relevance in the discussions dealing with timed machines, it is of fundamental importance mathematically, and so we introduced it at this early stage. Again, the notation has been chosen to aid the ensuing discussions, but also follows logically from standard notation. Additionally, the motivation behind many of the propositions was to fix the concepts and notation in mind, rather than in developing tools to be used later on.

Following this is a generic discussion of strings, relations, and then functions including a convention treating partial functions as total functions which will be used repeatedly. We also presented a variety of isomorphisms between functions, only some of which are to be used later on. The remainder provide the setting, and provide other natural ways to join two functions together to get a third.

The discussion of partitions is of particular importance to the question of decomposition, as will become clear in subsequent chapters, and hence particular attention should be paid it, especially the lattice of partitions.

The last sections of the chapter introduce the three sorts of abstract machines that we will be dealing with throughout. They are essentially short reviews of standard works on each sort of machine, with definitions and examples. The section on timed machines is considerably longer than the other two—in part because of their more complicated structural presentation, and in part because they are not as well covered in the literature, and hence are probably less known to the reader.

This brings to a close our discussion of the more general terms and concepts that are central to the study of the sorts of abstract machines we are looking at. Once more, we refer the reader to the appropriate texts for more complete discussions on all of these concepts. Our next chapter illustrates (and slightly extends) existing theory concerning the parallel decomposition of finite machines. This provides some answers to our original question of when a large machine may be broken up into components.

Chapter 3

Parallel Decomposition of Untimed Machines

This chapter is largely a review of appropriate sections of [Shi87] and extensions to the ω-machines of [Tho90]. [Shi87] deals with transducers, and we simplify the work slightly by looking at language acceptors. This does not affect the content to any great extent, as we are more concerned with the structure of the machine, which is concentrated in its edges, and there is less notation to obscure the simplicity of the results.

3.0.1 Parallel Composition of Untimed Machines

In an attempt to formalise the concept of a lock-step/parallel decomposition, we first discuss what we mean by a parallel composition. Each component will process the inputs independently and an input will be accepted only if each component accepts it, the set of inputs being common to all the machines in question.

Initially we look at the case where there are only two component machines in parallel and they have the same set of actions. We let $\mathcal{M}_1$ and $\mathcal{M}_2$ denote the two components and $\mathcal{M}_{||}$ their composition. We now try to determine specifications for $\mathcal{M}_{||}$ given the specifications for $\mathcal{M}_1$ and $\mathcal{M}_2$.

Since $\mathcal{M}_1$ and $\mathcal{M}_2$ can both be in any of their respective states, we let $Q_{||} = (Q_1, Q_2)$. Thus, the state set of $\mathcal{M}_{||}$ will consist of all the ordered pairs whose first component is from Q_1 and whose second component is from Q_2. The set of all possible actions for $\mathcal{M}_{||}$ will be the same as the set of actions for both $\mathcal{M}_1$ and $\mathcal{M}_2$. That is, $\Sigma_{||} = \Sigma_1 (= \Sigma_2)$.

In order to get each component of a parallel machine processing the input independently, an edge in the parallel machine will correspond to an edge in each component. Thus the edge $((q_1, q_2), a, (q_1', q_2'))$ will be in $E_{\|}$ precisely when (q_1, a, q_1') is in E_1 and (q_2, a, q_q') is in E_2. For similar reasons, we start the global machine with each component in its starting state, and so let $s_{\|} = (s_1, s_2)$.

There is no intuitive reason for any particular definition of the accepting states, but to keep the concept of the lock-step parallel composition as a Cartesian Product of component machines, and to ensure that an input in the parallel machine is accepted precisely when each component also accepts it, we define $F_{\|}$ to be (F_1, F_2) for finite machines, and $\mathcal{F}_{\|}$ to be $(\mathcal{F}_1, \mathcal{F}_2)$ for Muller machines. This completes the intuitive motivation for the following definition.

Definition 3.1

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two finite machines with the same alphabet. We define the parallel composition *of $\mathcal{M}_1$ and $\mathcal{M}_2$, $\mathcal{M}_1 \,\|\, \mathcal{M}_2$, to be the machine, $\mathcal{M}_{\|}$, where :*

- $Q_{\|} = Q_1 \times Q_2$
- $\Sigma_{\|} = \Sigma_1 (= \Sigma_2)$
- $E_{\|} \subseteq (Q_{\|}, \Sigma_{\|}, Q_{\|})$ *is defined as*

$$E_{\|} = \{((q_1, q_2), a, (q_1', q_2')) | (q_1, a, q_1') \in E_1 \text{ and } (q_2, a, q_2') \in E_2\}$$

- $s_{\|} = (s_1, s_2)$
- $F_{\|} = (F_1, F_2)$

The definition for the parallel composition of two Muller machines is the same as that for finite machines, excepting that the accepting sets become accepting collections of sets. The definition reinforces the idea of the parallel composition being a Cartesian Product.

Definition 3.2

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two Muller machines with the same alphabet. We define the parallel composition *of $\mathcal{M}_1$ and $\mathcal{M}_2$, $\mathcal{M}_1 \,\|\, \mathcal{M}_2$, to be the machine, $\mathcal{M}_{\|}$, where :*

- $Q_{\|}, \Sigma_{\|}, E_{\|}$ *and* $s_{\|}$ *are as in Definition 3.1.*
- $\mathcal{F}_{\|} = (\mathcal{F}_1, \mathcal{F}_2)$

Consider the example depicted in Figure 3.1. It will be referred to, and expanded upon throughout this chapter.

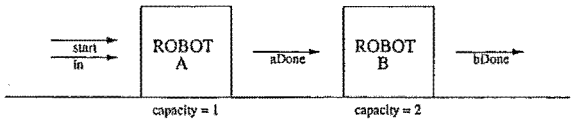

Figure 3.1: A simple production line

The picture above describes a simple production line consisting of two robots. An object to be worked on arrives at robot A, signified by an *in* move or a *start*. When robot A has finished working on the object, it is passed to robot B, signified by an *aDone* move. When robot B finishes working on the object it leaves the system, a *bDone* move.

Suppose Robot A has a workbench capacity of 1, whilst robot B has a workbench capacity of 2. Then we can describe the actions of each as an ω-machine (we use ω-machines rather than finite machines because it seems more intuitive that the system would not be designed to stop). For example, the two machines in Figure 3.2 would model their behaviours—note that actions whose transitions cause no change of state (and are always defined) are not shown, and all cycles are accepted.

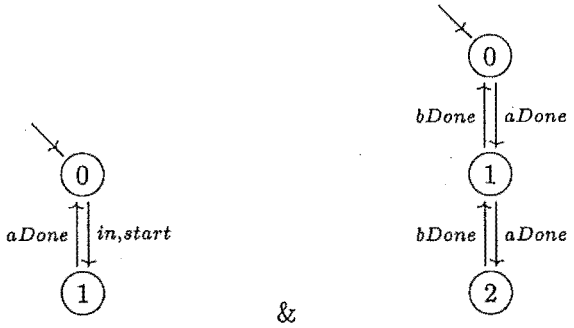

Figure 3.2: Robot A and Robot B as separate ω-machines.

If we let $\mathcal{M}_A$ and $\mathcal{M}_B$ be the ω-machines describing the behaviour of Robot A and B respectively then, using the above definition (Definition 3.2), we can form the parallel composition of the two machines, $\mathcal{M}_A \parallel \mathcal{M}_B$. It is the machine represented in Figure 3.3—noting that all cycles are accepted, and that a, b, and t represent "*aDone*", "*bDone*" and "*start*" respectively.

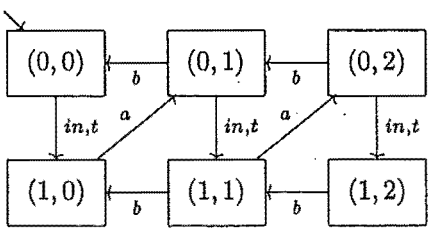

Figure 3.3: Pline—an example of parallel composition.

Observe that the parallel composition above does indeed model the production line as a whole, thus verifying that the definition does indeed satisfy some of the intuitions behind a parallel composition. Also observe the regularity that is present in the digraph. Some of these regularities we will formalise, and be using later.

The natural extensions to parallel compositions of many machines are as follows.

Definition 3.3

Let $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_n$ *be a collection of finite machines—we define their* parallel composition *to be the machine* $\mathcal{M}_{\|}$ *where :*

1. $Q_{\|} = (Q_1, Q_2, ..., Q_n)$
2. $\Sigma_{\|} = \Sigma_1 (= \Sigma_2 = ... = \Sigma_n)$
3. $E_{\|} \subseteq (Q_{\|}, \Sigma, Q_{\|})$ *is defined as*

$$E_{\|} = \{((q_1, q_2, ..., q_n), a, (q_1', q_2', ..., q_n')) \mid \forall i \in 1, 2, ..., n, (q_i, a, q_i') \in E_i\}$$

4. $s_{\|} = (s_1, s_2, ..., s_n)$
5. $F_{\|} = (F_1, F_2, ..., F_n)$

Definition 3.4

Let $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_n$ *be a collection of* ω*-machines—we define their* parallel composition *to be the machine* $\mathcal{M}_{\|}$ *where :*

1. $Q_{\|}, \Sigma_{\|}, E_{\|}, s_{\|}$ *are as in Definition 3.3.*
2. $\mathcal{F}_{\|} = (\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_n)$

In either case, the many component definition restricted to the case with just two components is equivalent to the two component definition.

Suppose for example, that the production line example as shown above was incomplete, and that, in addition to the two robots shown, there was some tester at the end of the production line that checks the final product. If it is satisfactory, then the object leaves the system, and if not, then gets passed back to robot A to work on. Thus the flow of an object through the system is as illustrated in Figure 3.4.

Suppose we have ω-machines whose behaviours represent each of the two robots and the tester. The robots will behave as before, but the descriptions on will have to change to allow it to process "*accept*"s

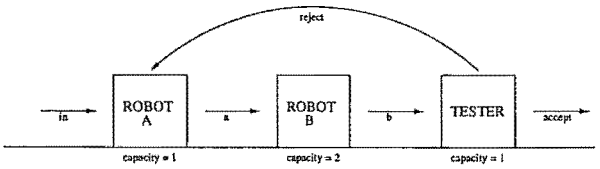

Figure 3.4: Simple factory (production line + tester)

and "*reject*"s. Tester will be as described below—noting that all cycles are acceptable, and we use $\surd$ to denote an acceptance, $\times$ for a rejection. Again we use the convention that actions whose transitions never "do" anything, are not shown.

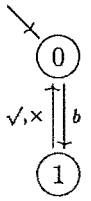

Figure 3.5: Tester as an ω-machine.

If, as before, the machines representing Robot A and Robot B are $\mathcal{M}_A$ and $\mathcal{M}_B$ respectively; and the machine representing Tester is $\mathcal{M}_T$ then we can, using Definition 3.4, define the parallel composition of the three, $\mathcal{M}_A \parallel \mathcal{M}_B \parallel \mathcal{M}_T$. This gives us the machine represented by Figure 3.6 and Figure 3.7 where again all cycles are accepted. We use both figures to describe the machine in order that the transition (edge) structure will be more transparent. Putting all the transitions on one picture causes the structure to be less evident.

We will call the parallel composition above "Factory". Thus we have now developed the following machines for consideration—Robot A, Robot B, Tester, Pline (= Robot A || Robot B), and now Factory (= Robot A || Robot B || Tester). Pline will be described by the machine $\mathcal{M}_P$ and Factory by $\mathcal{M}_F$. Once more, note the regularity present in the transition structure of Factory. This will be important in later discussions when we try to determine conditions for decomposition.

Although we will be restricting our attention initially to the two component case, we continue with the Factory example because it is large enough to make the structures present more identifiable. In fact, as we shall later see (Lemma 3.11), Factory is essentially "the same" as Pline || Tester anyway.

For either sort of machine determinism in the components is equivalent to determinism in the global machine. This is useful because, as discussed earlier, we only want to consider deterministic machines. We only give the result in the two component case, because the extension to the many component case

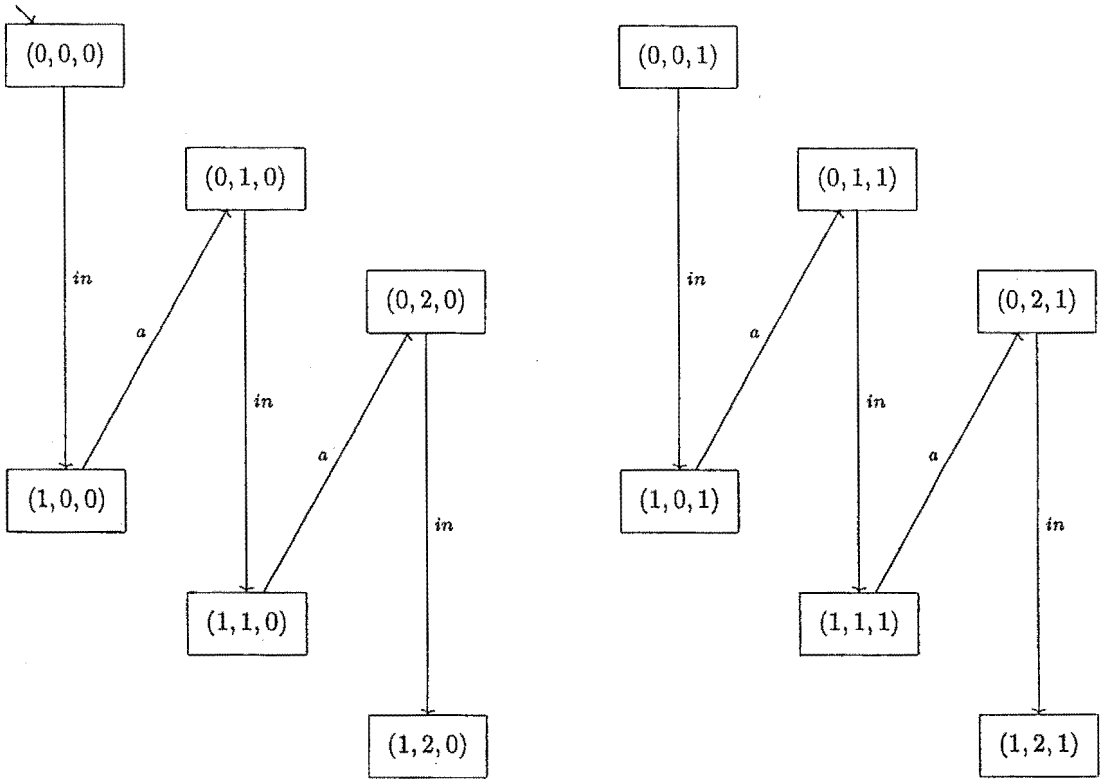

Figure 3.6: Factory—$\mathcal{M}_A \parallel \mathcal{M}_B \parallel \mathcal{M}_T$, only *in* and a transitions

is very natural, and the notation can be obscuring.

For this proof, and subsequent ones in this chapter - the theorem is stated for both Finite Machines and ω-automata, and the proofs follow through for both at the same time, excepting when we consider the last components (which differ in their specifications) and in these cases the pertinent part of the proof will be broken up. We join the theorems and proofs together, as it highlights the structural similarities between the two types of machines, and also because we feel no further illumination is to be gained from segregating the two cases. Keep in mind though, that there are really two collections of results being presented as one—and that the structures of the two types of machines are different.

Lemma 3.5

$\mathcal{M}_1 \parallel \mathcal{M}_2$ is deterministic iff both $\mathcal{M}_1$ and $\mathcal{M}_2$ are deterministic.

Proof:

If we let $\mathcal{M}_\parallel = \mathcal{M}_1 \parallel \mathcal{M}_2$ then saying that $\mathcal{M}_\parallel$ is deterministic is equivalent to saying that for each member of $Q_\parallel$, (q_1, q_2), and each input, a, there is a unique edge in $E_\parallel$ of the form $((q_1, q_2), a, (q_1', q_2'))$ for some $(q_1', q_2') \in Q_\parallel$.

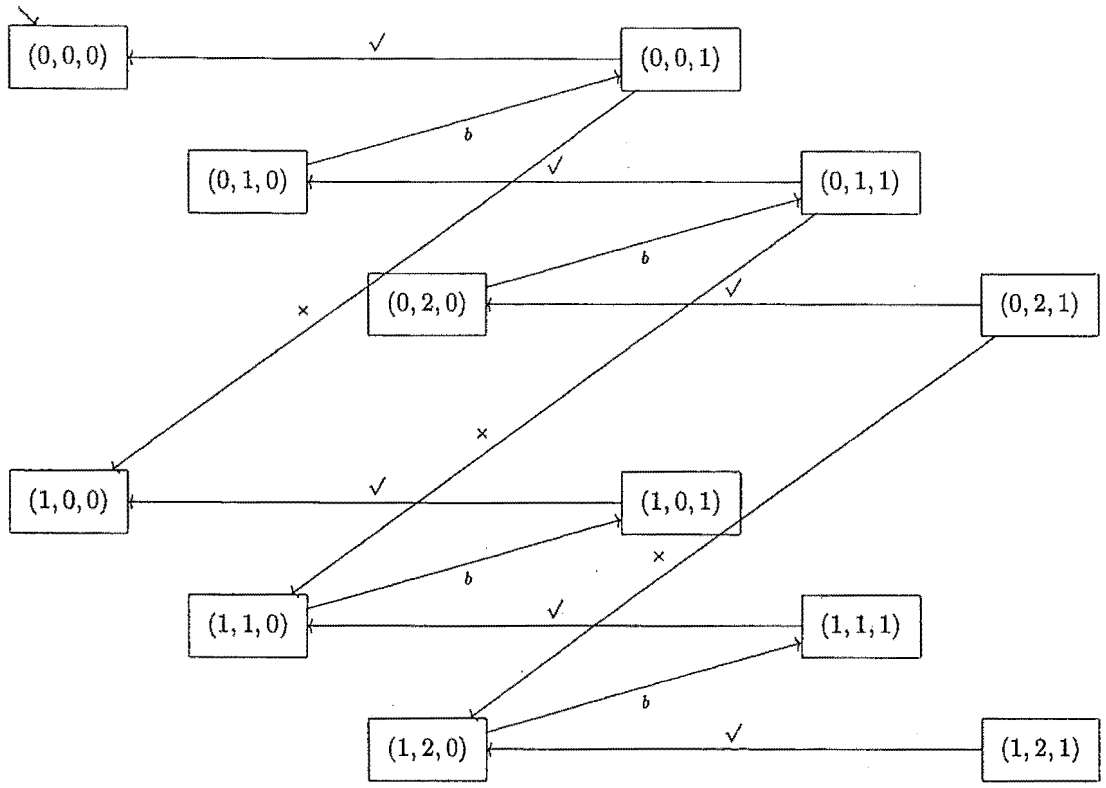

Figure 3.7: Factory—$\mathcal{M}_A \parallel \mathcal{M}_B \parallel \mathcal{M}_T$, only b, $\surd$ and $\times$ transitions

We note that $Q_{\parallel}$ is the whole of (Q_1, Q_2), and that $((q_1, q_2), a, (q_1', q_2'))$ is an edge of the parallel machine precisely when (q_1, a, q_1') is an edge in $\mathcal{M}_1$ and (q_2, a, q_2') is an edge in $\mathcal{M}_2$.

If we assume that the parallel machine is deterministic, but one of the component machines, WLOG $\mathcal{M}_1$, is not, then for some state, $q_1 \in Q_1$, and input, a, there are two distinct states $q_1' \in Q_1$ and $q_1'' \in Q_1$, so that both (q_1, a, q_1') and (q_1, a, q_1'') are edges of $\mathcal{M}_1$ or there is no $q_1' \in Q_1$ so that (q_1, a, q_1') is in $E_{\parallel}$.. If the latter is true or there is no edge of the form (q_2, a, q_2') in E_2 then this contradicts the fact that existence of an edge in the parallel machine. On the other hand, if there is some (q_2, a, q_2') in E_2 then this contradicts the uniqueness of the edge in the parallel machine.

For the converse if you assume that both component machines are deterministic then for each (q_1, q_2) in $Q_{\parallel}$ and input, a, there is a unique $q_1' \in Q_1$ and $q_2 \in Q_2$ so that $(q_1, a, q_1') \in E_1$ and $(q_2, a, q_2') \in E_2$. But this means $((q_1, q_2), a, (q_1', q_2')) \in E_{\parallel}$ (and is unique), so that $\mathcal{M}_{\parallel}$ is unique as required.

■□■

Notice that this guarantees that both Pline and Factory will be deterministic because of the determinacy of its components. Thus Figure 3.3, Figure 3.6 and Figure 3.7 are verifications of the above result.

Because each edge in a parallel machine corresponds to an edge in each of the component machines, we find from the above lemma that the transition functions correspond in a natural way. This result is used in a lot of the proofs that follow.

Lemma 3.6

If $\mathcal{M}_{\|} = \mathcal{M}_1 \,\|\, \mathcal{M}_2$ *then* $\Delta_{\|}[Q_{\|}, \Sigma] = (\Delta_1[Q_1, \Sigma], \Delta_2[Q_2, \Sigma])$

Note that this is equivalent to saying that for each state, (q_1, q_2), in $Q_{\|}$ and input, a,

$$\Delta_{\|}((q_1, q_2), a) = (\Delta_1(q_1, a), \Delta_2(q_2, a))$$

Proof:

From Lemma 3.5 we know that $\mathcal{M}_{\|}$ being deterministic implies that $\mathcal{M}_1$ and $\mathcal{M}_2$ are both deterministic as well.

Take an arbitrary $(q_1, q_2) \in Q_{\|}$ and input, a. Now if $\Delta((q_1, q_2), a) = (q_1', q_2')$ for some $(q_1', q_2') \in Q_{\|}$, then $((q_1, q_2), a, (q_1', q_2')) \in E_{\|}$.

By the definition of $E_{\|}$ this means that $(q_1, a, q_1') \in E_1$ and $(q_2, a, q_2') \in E_2$, so that $\Delta(q_1, a) = q_1'$ and $\Delta(q_2, a) = q_2'$. Combining these gives $\Delta_{\|}((q_1, q_2), a) = (\Delta_1(q_1, a), \Delta_2(q_2, a))$ as required.

■□■

Again, referring back to Factory, and Figure 3.6, note $\Delta_F((1, 0, 1), a) = (0, 1, 1)$ where Factory is represented by $\mathcal{M}_F$. The natural extension of the above result to many components would say that this $\Delta_A(1, a)$ must equal 0, $\Delta_B(0, a)$ must equal 1, and $\Delta_T(1, a)$ must equal 1 where $\mathcal{M}_A, \mathcal{M}_B, \mathcal{M}_T$ represent Robot A, Robot B, and Tester as mentioned in the discussion preceding Figure 3.6. Referring back to Figure 3.2, Figure 3.3 and Figure 3.5 we see that each of these transitions is indeed present. The other transitions are similar.

Although language relationships are not the focus of this work, we how here that composing machines yields language intersection. This shows that some of the desired properties of the parallel machine are indeed satisfied by the specifications as given here. The property demonstrated here is that an input in the parallel machine is accepted precisely when it is accepted by both component machines.

Lemma 3.7

$L(\mathcal{M}_1 \parallel \mathcal{M}_2) = L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.

Proof:

If $\alpha \in L(\mathcal{M}_1 \parallel \mathcal{M}_2)$ then there is an accepting run, $\sigma_\parallel \in Q_\parallel^\omega$, of α on $\mathcal{M}_\parallel$.

By definition of an accepting run, this means that the following are true :-

1. $\sigma_\parallel(0) = s_\parallel$
2. For each $i \in \omega$, $\sigma_\parallel(i) \xrightarrow{\alpha(i)} \sigma_\parallel(i+1)$
3. $In(\sigma_\parallel) \in \mathcal{F}_\parallel$

If we let $\sigma_1 \in Q_1^\omega$ be defined by $\sigma_1 = P_1 \circ \sigma_\parallel$ then the following hold :-

1. Because $\sigma_\parallel(0) = s_\parallel = (s_1, s_2)$ we have that

$$\begin{aligned} \sigma_1(0) &= (P_1 \circ \sigma_\parallel)(0) \\ &= s_1 \end{aligned}$$

2. For each $i \in \omega$, because $\sigma_\parallel(i) \xrightarrow{\alpha(i)} \sigma_\parallel(i+1)$, we know that

$$((P_1 \circ \sigma_\parallel)(i), (P_2 \circ \sigma_\parallel)(i)) \xrightarrow{\alpha(i)} ((P_1 \circ \sigma_\parallel)(i+1), (P_2 \circ \sigma_\parallel)(i+1))$$

   so that $\sigma_\parallel(i) \xrightarrow{\alpha(i)} \sigma_\parallel(i+1)$ from Lemma 3.6. This is equivalent to $\sigma_1(i) \xrightarrow{\alpha(i)} \sigma_1(i+1)$.

3. Because Q_2 is finite, and $In(\sigma_\parallel) \in \mathcal{F}_\parallel = (\mathcal{F}_1, \mathcal{F}_2)$, we know that $In(\sigma_1) = In(P_1 \circ \sigma_\parallel) \in \mathcal{F}_1$.

The above three conditions imply that σ_1 is an accepting run of α on $\mathcal{M}_1$, so that $\alpha \in L(\mathcal{M}_1)$.

A similar argument gives $\alpha \in L(\mathcal{M}_2)$, so that $\alpha \in L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$. α being an arbitrary member of $L(\mathcal{M}_1 \parallel \mathcal{M}_2)$, implies that $L(\mathcal{M}_1 \parallel \mathcal{M}_2) \subseteq L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$.

The argument for inclusion in the other direction is almost exactly the reverse of the above. Take accepting runs for α on each component machine, say σ_1 and σ_2. Define $\sigma_\parallel = \sigma_1 \oplus \sigma_2$ and it is just a matter of checking the definitions to see that $\sigma_\parallel$ defined in this way is an accepting run of α on $\mathcal{M}_\parallel$. α again being arbitrary implies the opposite inclusion.

Thus $L(\mathcal{M}_1 \parallel \mathcal{M}_2) = L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$ as required.

■□■

Unfortunately, the languages of factory and its components are too complicated for us to describe them here in full detail. For example, $L(\mathcal{M}_A)$ is equal to

$$((b+\surd)^*(in+x)(b+\surd)^*a)^*((b+\surd)^\omega + (b+\surd)^*(in+x)(b+\surd)^\omega) + ((b+\surd)^*(in+x)(b+\surd)^*a)^\omega$$

and the languages for each of the other machines are at least as complicated—with their intersections even more complicated. However, the intersection property still holds. To illustrate this, consider a sequence accepted by $\mathcal{M}_A$, but not by $\mathcal{M}_B$, for example b^ω. Check that b^ω is not accepted by either $\mathcal{M}_P$ or $\mathcal{M}_F$. This is indeed the case. Also consider a sequence accepted by $\mathcal{M}_F$, for example $\alpha = (in.a.b.\surd)^\omega$, and check that this is accepted by any of the component machines. This provides some empirical support for the previous result.

3.1 Comparing Untimed Machines

In order to describe the relationships between machines, we introduce a series of notions of equivalence and comparison, between pairs of machine. Ideally, we would like them to correspond very naturally with any intuitive notions we have. It will normally be the case that a map between machines, $\phi : \mathcal{M}_1 \to \mathcal{M}_2$, will be determined completely by the type of map and $\phi|_{Q_1}$, the map restricted to the states. When this is true we interchange freely $\phi : \mathcal{M}_1 \to \mathcal{M}_2$ with $\phi : Q_1 \to Q_2$, relying on context to distinguish between the two.

The first of these notions is equality. The definition is follows from the definition of a machine as an ordered pair.

Definition 3.8

Two untimed machines, $\mathcal{M}_1$ and $\mathcal{M}_2$, are regarded as equal *if the components of each machines are equal. That is, the machines must be the same type, and if they are finite machines then $Q_1 = Q_2, \Sigma_1 = \Sigma_2, f_1 = f_2, s_1 = s_2$, and $F_1 = F_2$. ω-machines are the same same except the $F_1 = F_2$ requirement is replaced by $\mathcal{F}_1 = \mathcal{F}_2$.*

We write $\mathcal{M}_1 = \mathcal{M}_2$ if $\mathcal{M}_1$ and $\mathcal{M}_2$ are equal. Despite simplicity, we wish to discuss more abstract structural relationships. Just comparing things on the basis of equality does not allow for any simplifications in the analysis of the behaviour of a machine. Other notions of equivalence, or comparison, are less obvious and are perhaps best motivated by means of example.

To see more clearly how equality is too restrictive a notion we look at how far it takes us in answering the sorts of questions we are interested in here. That is, when can we break a large machine up into

smaller components. If we require that the large machine be *equal* to the parallel composition of the smaller components, then its states must *already* be described as ordered pairs, etc. Indeed there is no real analysis or simplification to be had.

For example, Factory as described earlier is the parallel composition of Robot A, Robot B, and Tester. Because Pline is the parallel composition of Robot A and Robot B, we would also like to think of Factory as being the parallel composition of Pline and Tester. But it is not the case that the two are equal since the states of Factory are ordered triplets whilst the states of $\mathcal{M}_P \parallel \mathcal{M}_T$ are ordered pairs. So, if a large machine's states were not described as ordered pairs then it would not be possible to view this machine as a parallel composition of smaller machine. Thus equality is too strict a criterion for our consideration.

3.1.1 Isomorphisms between Untimed Machines

The use of the word isomorphism is prevalent in mathematics, and typically means a structure preserving relabelling (iso = same, morph = shape or structure, ism = process). We use it here in much the same fashion. Consider the machines defined by the following pictures.

Figure 3.8: Example of similar finite machines.

We also describe a somewhat more pertinent pair of ω-machines who we would like to describe as being the same. Earlier we defined Pline as the parallel composition of Robot A and Robot B. Suppose we define Oline as the parallel composition of Robot B and Robot A, described by the machine $\mathcal{M}_O$. Then Oline would be described by the following picture where once more transitions from unimportant actions are excluded, and all cycles are accepted.

We would like to describe Oline and Pline as being the same. However, they are not equal, as discussed earlier ($Q_B \times Q_A \neq Q_A \times Q_B$). Our intuitions also tell us that a parallel machine should be independent of the order in which its components are listed.

In both of the above examples, although the machines do not satisfy the definition of equality, it is "obvious" that the only differences are the superficial labels on the vertices. The structure of the two machines is the same. Thus we say machines are isomorphic if their states correspond in a structure preserving manner. We require that the actions remain unchanged, which is non-intuitive when we view

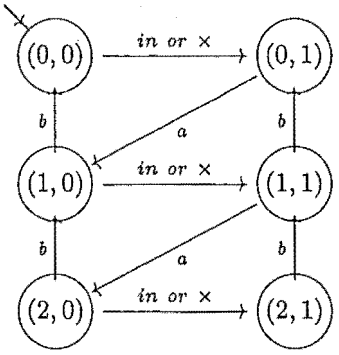

Figure 3.9: Oline = Robot B || Robot A

the actions merely as labels on the structure, rather than as a part of the structure itself. A more natural approach would be to allow the actions on transitions to be relabelled as well.

This is not the approach we choose, however, for two reasons. The situations we are attempting to describe using abstract machines involve interactions with the outside world, represented as transitions, which are important enough structurally that we do not wish to treat machines which perform different actions the "same". The second reason, tied in with the first, is that an alternative way of viewing a deterministic machine (and in many cases a much more useful one), has the action set as an index, and a whole range of transition functions, one for each action, so that

$$\mathcal{M} = (Q, \Sigma, \{\Delta_i\}_{i \in \Sigma}, s, F)$$

where each $\Delta_i : Q \to Q$, and $\Delta_a(q) = q'$ would be equivalent to $(q, a, q') \in E$ in the conventional description. If we look at a machine this way then relabelling actions makes less sense.

At any rate, the approach we take allows a relabelling of the states. That is, for two machines to be considered isomorphic there must be some map, ϕ, from the state set of one to the state set of the other.

The structure preservation will require that the map be bijective (so that each state in the one machine will correspond to exactly one state in the other machine) and that if (q, a, q') is an edge in the first machine, then $(\phi(q), a, \phi(q'))$ will be an edge in the second machine. Similarly, starting states will correspond, so that if s is the starting state of the first machine, then $\phi(s)$ will be the starting state of the second machine. Final states/sets will similarly correspond, motivating the following definition.

Definition 3.9

Given two finite machines, $\mathcal{M}_1$ and $\mathcal{M}_2$, we describe them as being isomorphic *if $\Sigma_1 = \Sigma_2$ and there is a bijective map $\phi \in Q_2^{Q_1}$:*

1. $q \xrightarrow{a} r$ *if and only if* $\phi(q) \xrightarrow{a} \phi(r)$.

2. $s_2 = \phi(s_1)$, *and*

3. $F_2 = \phi[F_1]$.

The definition for ω-machines is the same except that the last requirement is replaced by $\mathcal{F}_2 = \phi[\mathcal{F}_1]$. The function ϕ is called an isomorphism *from $\mathcal{M}_1$ to $\mathcal{M}_2$.*

We write $\mathcal{M}_1 \equiv \mathcal{M}_2$ if $\mathcal{M}_1$ is isomorphic to $\mathcal{M}_2$, and write $\phi : \mathcal{M}_1 \equiv \mathcal{M}_2$ to denote that ϕ is an isomorphism from $\mathcal{M}_1$ to $\mathcal{M}_2$.

By Definition 3.9, the two machines in the Figure 3.8 are isomorphic, with $\phi : \mathcal{M}_1 \equiv \mathcal{M}_2$ begin defined by $\phi = \{(1, A), (2, B), (3, C), (4, D)\}$. Similarly we see that Pline and Oline are isomorphic, with $\phi : \mathcal{M}_P \equiv \mathcal{M}_O$ defined by $\phi(x, y) = (y, x)$ for each $(x, y) \in Q_P$.

Because an isomorphism, ϕ, is a bijection, it is invertible. That is, ϕ^{-1} is a total function, and is in fact an isomorphism. Composing ϕ and ϕ^{-1} gives another isomorphism, the identity isomorphism. The identity isomorphism, $Id_{\mathcal{M}} : \mathcal{M} \equiv \mathcal{M}$ is defined by $Id_{\mathcal{M}}(x) = x$ for each $x \in Q$.

Proposition 3.10 (Shi87)

If $\phi : \mathcal{M}_1 \equiv \mathcal{M}_2$ then $\phi \circ \phi^{-1} = Id_{\mathcal{M}_1}$ and $\phi^{-1} \circ \phi = Id_{\mathcal{M}_2}$.

Thus we know that equality is a stricter condition than isomorphism, since $Id_{\mathcal{M}} : \mathcal{M} \equiv \mathcal{M}$.

How does the introduction of isomorphism help us to answer our original question? We want to see when we can break a machine up into parallel components. As discussed earlier our intuition tells us that it shouldn't matter which order the components are in—the parallel machine is "the same". They are indeed isomorphic, which means they have the same structure. That is, Pline and Oline being isomorphic is representative of a general result for parallel machines.

Lemma 3.11 (Shi87)

If $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$ are finite machines, then :

- $\mathcal{M}_1 \parallel \mathcal{M}_2 \equiv \mathcal{M}_2 \parallel \mathcal{M}_1$, *and*
- $\mathcal{M}_1 \| (\mathcal{M}_2 \parallel \mathcal{M}_3) \equiv (\mathcal{M}_1 \parallel \mathcal{M}_2) \parallel \mathcal{M}_3$.
- $\mathcal{M}_1 \| (\mathcal{M}_2 \parallel \mathcal{M}_3) \equiv \mathcal{M}_1 \parallel \mathcal{M}_2 \parallel \mathcal{M}_3$.

Proof:

For the first result, define $\phi : (Q_1, Q_2) \to (Q_2, Q_1)$ be defined by $\phi(q_1, q_2) = (q_2, q_1)$ for all $(q_1, q_2) \in (Q_1, Q_2)$, and then verify that this ϕ does indeed satisfy the requirements of an isomorphism. The second is the same with $\phi : (Q_1, (Q_2, Q_3)) \to ((Q_1, Q_2), Q_3)$ defined by $\phi(q_1, (q_2, q_3)) = ((q_1, (q_2, q_3))$ for each $(q_1, (q_2, q_3)) \in (Q_1, (Q_2, Q_3))$. The third is similar.

■□■

In addition to telling us that Pline and Oline are isomorphic, the above result also tells us that Factory is isomorphic to Pline || Tester, which is why we can use Factory as an example of a machine with two components.

Just a quick note that the language behaviour of isomorphic machines reflects the intuition of structural equivalence. That is, the behaviours of isomorphic machines are identical. Observe that this is the case for the isomorphic machines in Figure 3.8, as well as for Pline and Oline.

Proposition 3.12 (Shi87)

If $\mathcal{M}_1$ and $\mathcal{M}_2$ are isomorphic then $L(\mathcal{M}_1) = L(\mathcal{M}_2)$.

It is easy to verify that the composition of two isomorphisms is itself an isomorphism, and thus that $\equiv$ is an equivalence relation on any set of finite machines. This, coupled with Lemma 3.11, means that the structure of a parallel machine is independent of the order of its components, or whether we regard some of its components themselves as parallel compositions. Thus Factory $\equiv$ Tester || Oline $\equiv$ Robot A || Tester || Robot B.

This is better, using the concept of isomorphism we are now capable of comparing machines that are not identical. But in order to decompose a large machine into smaller components which combine to form a parallel machine, we need to develop further ways of comparing machines. As a demonstration of this, because the number of states of a large machine is the product of the number of states of each component—it would be impossible to decompose a machine with a prime number of states into components if we required the two machines to be isomorphic.

3.1.2 Homomorphisms between Untimed Machines

Using isomorphisms we now have a way of formally describing two machine as being structurally "the same", we now generalise this concept to give a way of describing one machines as being structurally "contained in" another. Once more, we will use an example to motivate the formal definition.

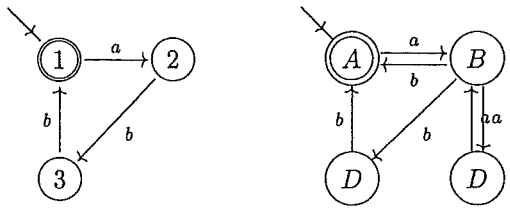

Figure 3.10: Example of one machine structurally contained in another.

Again, looking at the above example one can understand that it is natural to describe the machine on the left as being structurally contained in the machine on the right. The machine on the right has an additional state, and also some extra edges. Thus, the transition structure of the machine on the right is "greater" than that of the machine on the left. Indeed the machine on the left is isomorphic to a submachine of the machine on the right. We formalise this below.

Definition 3.13

Given two untimed machines, $\mathcal{M}_1$ and $\mathcal{M}_2$, we describe $\mathcal{M}_1$ as being homomorphic *to $\mathcal{M}_2$ if $\Sigma_1 = \Sigma_2$ and we can find a map $\phi : Q_1 \to Q_2$ so that:*

- *If* $q \xrightarrow{a} r$ *then* $\phi(q) \xrightarrow{a} \phi(r)$,
- $\phi(s_1) = s_2$, *and*
- $\phi[F_1] \subseteq F_2$.

The map ϕ is called a homomorphism. *If, in addition to being a homomorphism, ϕ is also injective (so that $q \neq r$ implies that $\phi(q) \neq \phi(r)$), then we say $\mathcal{M}_1$ is* epimorphic *to $\mathcal{M}_2$, and call ϕ an* epimorphism, *and write $\phi : \mathcal{M}_1 \leq \mathcal{M}_2$ (just $\mathcal{M}_1 \leq \mathcal{M}_2$ if ϕ is not given).*

In [Shi87], if $\mathcal{M}_1$ is homomorphic to $\mathcal{M}_2$ then he describes $\mathcal{M}_1$ as being implemented by $\mathcal{M}_2$; if $\phi : \mathcal{M}_1 \leq \mathcal{M}_2$ then $\mathcal{M}_1$ is realized by $\mathcal{M}_2$ and ϕ is called a state behaviour assignment. We use epimorphism and use the above conjugations to be consistent with standard mathematical literature concerning structural relationships of abstract objects. The injective-ness of the homomorphism ensures that $\mathcal{M}_1$ is isomorphic to a submachine of $\mathcal{M}_2$. Because our concept of structural equivalence is the existence of an isomorphism, our concept of structural inclusion is the existence of an epimorphism.

This means that the machines in Figure 3.10 are epimorphic. That is, if they are described by $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively from left to right then $\mathcal{M}_1 \leq \mathcal{M}_2$.

Just before going on, just observe that none of Robot A, Robot B, or Tester are epimorphic (or indeed homomorphic) to Factory. The structural relationship between components and their composition is

more subtle than mere inclusion.

Now, from the definitions above one can see that any isomorphism must be a homomorphism, and a homomorphism must be an epimorphism. In fact, an isomorphism is just an invertible homomorphism (that is, a homomorphism with a homomorphism as an inverse).

The first requirement of a homomorphism means that any edge in the original machine will have a corresponding edge in the image machine. This carries over naturally to the transition function, in a manner similar to that of Lemma 3.6.

Lemma 3.14

If $\phi : \mathcal{M}_1 \leq \mathcal{M}_2$ *where* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *are both deterministic, then for each* $q \in Q_1$ *and* $a \in \Sigma$

$$\phi(\Delta_1(q,a)) = \Delta_2(\phi(q),a)$$

Proof:

For an arbitrary $q \in Q$ and $a \in \Sigma$, let $q' = \Delta_1(q,a)$.

Then $(q,a,q') \in E_1$ so that $(\phi(q),a,\phi(q')) \in E_2$ (by Definition 3.13). Thus, $\Delta_2(\phi(q),a) = \phi(q') = \phi(\Delta_1(q,a))$ as required.

■□■

For example, referring back to Figure 3.10, observe that $2 \xrightarrow{b} 3$ for $\mathcal{M}_1$, the machine on the left. $\phi : \mathcal{M}_1 \leq \mathcal{M}_2$ defined by $\phi = \{(1,A),(2,B),(3,C)\}$ is an epimorphism where $\mathcal{M}_2$ is the machine on the left. The above result say that for $\mathcal{M}_2$, $\phi(2) \xrightarrow{a} \phi(3)$. That is $B \xrightarrow{a} C$ must be a transition for $\mathcal{M}_2$. This is indeed the case.

Also the composition of two morphisms of any type is a morphism of the same type (where the morphisms could be isomorphisms, homomorphisms, or epimorphisms) so that an arbitrary collection of machines, along with any of the types of morphisms discussed here, will form a category.

We show the language relationship induced by an epimorphism (actually holds for homomorphism as well), to demonstrate ow the definition satisfies our intuition of structural inclusion.

Lemma 3.15

If $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *are either finite machines or* ω*-machines, and* $\mathcal{M}_1 \leq \mathcal{M}_2$ *then* $L(\mathcal{M}_1) \subseteq L(\mathcal{M}_2)$

Proof:

Take an arbitrary $\alpha \in L(\mathcal{M}_1)$. Then there exists a successful run, σ_1, of α on $\mathcal{M}_1$.

Now, if the machines in question are finite machines, then this means that

1. $\sigma_1(0) = s_1$
2. $\sigma_1(i) \xrightarrow{\alpha(i)} \sigma_1(i+1)$ for each $i \in \{0, 1, ..., n-1\}$
3. $\sigma_1(n) \in F_1$

Similarly, if the machines are ω-machines, then this means that

1. $\sigma_1(0) = s_1$
2. $\sigma_1(i) \xrightarrow{\alpha(i)} \sigma_1(i+1)$ for each $i \in \omega$
3. $In(\sigma_1) \in \mathcal{F}_1$

In either case, take an arbitrary $\phi : \mathcal{M}_1 \leq \mathcal{M}_2$ and let $\sigma_2 = \phi(\sigma)$. Then

1. $\sigma_2(0) = \phi(\sigma_1(0)) = \phi(s_1) = s_2$
2. For all appropriate i, that is $0 \leq i < sup(Range(\sigma_1))$, we have that $\sigma_1(i) \xrightarrow{\alpha(i)} \sigma_1(i+1)$, so that $\phi(\sigma_1(i)) \xrightarrow{\alpha(i)} \phi(\sigma_1(i+1))$. That is, $\sigma_2(i) \xrightarrow{\alpha(i)} \sigma_2(i+1)$.
3. If the machines in question are finite machines, then $\sigma_2(n) = \phi(\sigma_1(n)) \in \phi[F_1] \subseteq F_2$. If, on the other hand, the machine in question are ω-machines, then $In(\sigma_2) = In(\phi(\sigma_1)) = \phi(In(\sigma_1)) \subseteq \phi[\mathcal{F}] \subseteq \mathcal{F}_2$—the last equality holding by virtue of the finiteness of the state spaces.

Thus in either case, σ_2 is a successful run of α on $\mathcal{M}_2$, so that $\alpha \in L(\mathcal{M}_2)$. α being arbitrary completes the proof.

■□■

Thus $w = (a.b.b)^*$ is accepted by the machine on the right of Figure 3.10. It being epimorphic to the machine on the right means that w must also be accepted by the machine on the right, which it is, verifying the result.

3.2 Parallel Decomposition of Untimed Machines

This is the focus of the whole chapter. The introduction of the earlier notation provides us with the tools necessary to formulate our original question. That is, if we have a large machine, can we find

smaller components, whose operation in parallel is at least that of the larger machine. We wish to examine the case in its full generality—with an arbitrary number of components — but as we shall later see Theorem 3.31 studying the two component case is sufficient. The tools to see this, however, are developed during the examination of the two component case.

We first formalise the question. If we are given a machine $\mathcal{M}$, we wish to find $\mathcal{M}_1$ and $\mathcal{M}_2$ (or possibly a larger collection of machines), so that $|Q_1|, |Q_2| < |Q|$ and $\mathcal{M} \leq \mathcal{M}_1 \parallel \mathcal{M}_2 = \mathcal{M}_{\parallel}$.

Definition 3.16

Let $\mathcal{M}$ be a finite machine or an ω-machine. We say that $\mathcal{M}$ has a parallel decomposition *if there is a collection of machines, each of whose state set is smaller that that of $\mathcal{M}$'s, so that $\mathcal{M}$ is epimorphic to their parallel composition.*

From our ongoing example, we see that Factory and Pline both have parallel decompositions (trivially, since they are each equal to a parallel composition of machines).

With the earlier discussion in mind, we have the following definition.

Definition 3.17

Let $\mathcal{M}$ be a finite machine or an ω-machine. We say that $\mathcal{M}$ has a bi-parallel decomposition *if there are two machines, $\mathcal{M}_1$ and $\mathcal{M}_2$, satisfying the following properties :*

1. $|Q_1|, |Q_2| < |Q|$
2. $\mathcal{M} \leq \mathcal{M}_1 \parallel \mathcal{M}_2$.

We write $\mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$ if $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfy the hypotheses above. If, in addition, we are also given $\phi : \mathcal{M} \leq \mathcal{M}_1 \parallel \mathcal{M}_2$ then we write $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$.

Pline, as described in Figure 3.3, has a bi-parallel decomposition. In fact, Factory also has a bi-parallel decomposition because it is epimorphic to the parallel composition of Pline and Tester. As we shall later see, this is representative of the general instance.

3.2.1 Necessary conditions for parallel decomposition

Although, Factory and Pline are special cases of large machines in that they are *isomorphic* to the parallel composition of smaller machines (rather than just epimorphic), we use them to motivate the results for the more general cases. In 3.2.4 we show how the results developed here can be applied to a machine which is not isomorphic to a parallel composition of some machine.

Factory and Pline were defined as parallel compositions, but we would like to see how to recover the components from the larger machines, and so we perform a balancing act—in part pretending we don't know that they are parallel compositions, in part acknowledging that they are.

The first thing to check before we attempt to answer when $\mathcal{M}$ has a parallel decomposition is to check to see if it might be possible to do it by mixing differing types of machines. It is obvious that the answer is no, since the behaviour of a finite machine is finite, and that of an ω-machine is infinite. This is imperative enough to the ensuing discussion that we state it here.

Proposition 3.18

If $\mathcal{M}$ is a finite machine, and $\mathcal{M}'$ is an ω-machine, then $L(\mathcal{M}) \cap L(\mathcal{M}') = \emptyset$.

Corollary 3.19

If $\mathcal{M}$ has a parallel decomposition, then the parallel decomposition is with machines of the same type as $\mathcal{M}$.

We use the regularities common to all parallel machines, along with the structural containment of $\mathcal{M}$ in $\mathcal{M}_1 \parallel \mathcal{M}_2$ to show that $\mathcal{M}$ must also have certain regularities.

Consider Pline along with Robot A. How can we recover Robot A, or at least its structure, from Pline? Consider what it "means" from the point of view of Pline when we say that Robot A is in state 0. What it means is that Pline must be in one of the states $(0,0)$, $(0,1)$, or $(0,2)$, but knowing just that Robot A is in state 0 does not enable us to decide which of these three states Pline is actually in. Thus state 0 for Robot A in a sense "corresponds" to each of states $(0,0)$, $(0,1)$, and $(0,2)$ in Pline. Similarly, state 1 in Robot A corresponds to each of states $(1,0)$, $(1,1)$, and $(1,2)$.

If we group these states together, we notice that the regularity in the transition structure of Pline becomes evident—a "b" transition causes Pline to remain in the same group; an "*in* or $\times$" transition causes a move from the first group to the second, whilst an "a" transition causes a move from the second group to the first. This is described by the following picture.

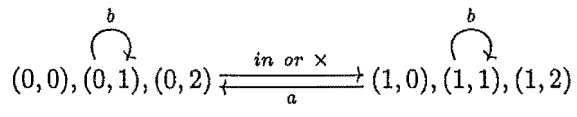

Figure 3.11: Grouping together appropriate states in Pline

This picture—ignoring the "b" transitions because they have no effect—has precisely the same transition structure as Robot A. This gives a way of generating the structure of Robot A from Pline (and the appropriate grouping of course).

If we assume for this discussion that we (acknowledge that we) have a bi-parallel decomposition of Pline. That is, we know Pline is structurally contained in Robot A || Robot B, and we also have the pertinent epimorphism—in this case the identity map. We can now generalise the process used to generate the above grouping of the states. For each state of Robot A we grouped together states from Pline which corresponded to that state.

More generally, if $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$, some obvious partitions (groupings) to be looking at would be those induced in a similar manner. For each $q_1 \in Q_1$, look at all the states, q, of $\mathcal{M}$ satisfying $P^1_\phi(q) = q_1$. That is, for each $q_1 \in Q_1$, consider $\{q \mid P^1_\phi(q) = q_1\}$. This set is just $\phi^{-1}[q_1, Q_2]$. Thus we have the partition $\{\phi^{-1}[q_1, Q_2] | q_1 \in Q_1\}$ which is $\phi^{-1}[Q_1, \top_{Q_2}]$.

As mentioned above, the motivation for this definition is that each component of $\mathcal{M}_\parallel$ will generate a partition of the states of $\mathcal{M}$ with this property. Since these generated partitions will be used later on (Lemma 3.22,Lemma 3.23, and Lemma 3.24, we define them outside of the proofs.

Definition 3.20

If $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$ then the natural partitions *of $\mathcal{M}$ (associated with ϕ) are $\phi^{-1}[Q_1, \top_{Q_2}]$ and $\phi^{-1}[\top_{Q_1}, Q_2]$.*

As an additional example—given the bi-parallel decomposition of Factory into Pline || Tester with epimorphism ϕ defined by $\phi(x, y, z) = ((x, y), z)$, the natural partition induced by Tester identifies all the states with 0 as their last component, and also all the states with 1 as their last component. Looking at Figure 3.6 or Figure 3.7 this partition splits the states left from right.

Notice that in both this case and the one preceding the definition, the transition structures generated by these partitions was deterministic. This is a requirement, because we are only considering deterministic partitions.

The states of $\mathcal{M}_\parallel$ are (Q_1, Q_2). Now, because $\mathcal{M}_\parallel$ realises $\mathcal{M}$, we know that $\mathcal{M}$ is structurally contained in $\mathcal{M}_\parallel$, by which we mean that $\mathcal{M}$ is isomorphic to some submachine of $\mathcal{M}_\parallel$. Because $\mathcal{M}_1$ is deterministic, and is the "projection" of $\mathcal{M}_\parallel$ onto it's first component, we find the states of $\mathcal{M}$ that correspond to states in $\mathcal{M}_\parallel$ with the same first component, must have the following property.

Definition 3.21

Given a machine, $\mathcal{M}$, and a partition of its states, π, then we say that π is an SP partition *if $\Delta[\pi, \Sigma] \leq \pi$ where $\leq$ is the partial ordering on sets of sets defined in Definition 2.5.*

The appellation of SP partition is from [Shi87]. Note that the definition is equivalent to saying that for each input and each member of π, the application of Δ to that member and input is contained within

another (not necessarily distinct) member of π. This alternative definition is the one used in [Shi87], but the one above has the advantage of brevity.

This now gives us the following Lemma.

Lemma 3.22 (Shi87)

The natural partitions of a machine are SP partitions.

Proof:

Let $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$ be the map generating the natural partitions. Let $\pi = \phi^{-1}[Q_1, \top_{Q_2}]$. Now $V \in \pi$ is equivalent to $V = \phi^{-1}[q_1, Q_2]$ for some $q_1 \in Q_1$.

Take an arbitrary $U \in \Delta[\pi, \Sigma]$. Then $U = \Delta[V, a]$ for some $V \in \pi$ and $a \in \Sigma$. Now, define $q_1 \in Q_1$ so that $V = \phi^{-1}[q_1, Q_2]$. Letting $q_1' = \Delta_1(q_1, a) \in Q_1$ gives the following.

$$\begin{aligned}
\phi[U] &= \phi[\Delta[V, a]] \\
&= \Delta_{\parallel}[\phi[V], a] && \textit{by Definition } 3.1 \\
&= \Delta_{\parallel}[(q_1, Q_2), a] \\
&= (\Delta_1(q_1, a), \Delta_2[Q_2, a]) && \textit{by Lemma } 3.6 \\
&\subseteq (q_1', Q_2) \\
&\in \pi
\end{aligned}$$

Because U was an arbitrary member of $\Delta[\pi, \Sigma]$ we have $\Delta[\pi, \Sigma] \leq \pi$ so that $\phi^{-1}[Q_1, \top_{Q_2}]$ is an SP partition. The argument for $\phi^{-1}[\top_{Q_1}, Q_2]$ is symmetric, giving the result.

■□■

Now consider Pline along with its natural partitions (assuming once more the "obvious" decomposition, $\phi : \mathcal{M}_P \trianglelefteq \mathcal{M}_A \| \mathcal{M}_B$). The natural partitions are given by

$$\pi_A = \{\{(0,0),(0,1),(0,2)\},\{(1,0),(1,1),(1,2)\}\}$$

and

$$\pi_B = \{\{(0,0),(1,0)\},\{(0,1),(1,1)\},\{(0,2),(1,2)\}\}.$$

Observe that for an arbitrary member of π_A, U, and an arbitrary member of π_B, V, that their intersection, $U \cap V$ is a singleton. In fact, because U must be of the form (q_A, Q_B) for some $q_A \in Q_A$ and V must be of the form (Q_A, q_B) for some $q_B \in Q_B$, $U \cap V$ will consist of precisely (q_A, q_B). Because U and V were

both arbitrary this means that π_A and π_B are orthogonal.

Taking account of structural inclusion (as opposed to equality), gives us the following result, which is really a consequence of the injective-ness of the map ϕ, just as Lemma 3.22 is really a consequence of the homomorphic nature of the map.

Lemma 3.23 (Shi87)

The natural partitions of a machine are orthogonal.

Proof:

Let $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$ be the map generating the natural partitions.

Let $\pi_1 = \phi^{-1}[Q_1, \top_{Q_2}]$ and $\pi_2 = \phi^{-1}[\top_{Q_1}, Q_2]$. Let W be an arbitrary member of $\pi_1.\pi_2$, so that $W = U \cap V$ for some $U \in \pi_1$ and $V \in \pi_2$.

But $U = \phi^{-1}[q_1, Q_2]$ for some $q_1 \in Q_1$ and $V = \phi^{-1}[Q_1, q_2]$ for some $q_2 \in Q_2$, so that

$$\begin{aligned} W &= U \cap V \\ &= \phi^{-1}[q_1, Q_2] \cap \phi^{-1}[Q_1, q_2] \\ &= \phi^{-1}[(q_1, Q_2) \cap (Q_1, q_2)] \\ &= \phi^{-1}[\{(q_1, q_2)\}] \end{aligned}$$

which is a singleton since ϕ is injective. Because W was arbitrary, we have that $\pi_1.\pi_2 = \bot_Q$ so that the natural partitions are orthogonal as required.

■□■

As an example the natural partitions of Factory induced by the obvious $\phi : \mathcal{M}_F \trianglelefteq \mathcal{M}_P \parallel \mathcal{M}_T$ are given by

$$\pi_T = \{\{(a, b, t) \mid (a, b) \in Q_P\} \mid t \in Q_T\}$$

and

$$\pi_P = \{\{(a, b, t) \mid t \in Q_T\} \mid (a, b) \in Q_P\}$$

and they are indeed orthogonal.

Just as each of the other requirements for a parallel decomposition (the homomorphism, and its injective-ness) gives a requirement on the induced partitions, so the state-size requirement forces them to be

non-trivial.

Lemma 3.24 (Shi87)

The natural partitions of a machine are non-trivial.

Proof:

Let $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_1 \parallel \mathcal{M}_2$ be the map generating the natural partitions.

If we assume one of the natural partitions is non-trivial, then because they are orthogonal, at least one of them must be $\perp_Q$. WLOG, Let $\phi^{-1}[Q_1, \top_{Q_2}] = \perp_Q$. Then,

$$\begin{aligned} |Q| &= |\perp_Q| \\ &= \phi^{-1}[Q_1, \top_{Q_2}] \\ &= |(Q_1, \top_{Q_2})| \quad \textit{since } \phi \textit{ is } 1-1 \\ &= |Q_1||\top_{Q_2}| \\ &= |Q_1| \end{aligned}$$

contradicting the assumption that $|Q_1| < |Q|$. Thus both natural partitions must be non-trivial as required.

■□■

Notice that π_A, π_B, π_P, and π_T as described earlier are all non-trivial.

Combining the above three results gives the following theorem.

Theorem 3.25 (Shi87)

If a machine has a bi-parallel decomposition, then it must have two non-trivial orthogonal SP partitions of its states.

Proof:

By Lemma 3.22, Lemma 3.23, and Lemma 3.24 the natural partitions satisfy the criteria.

■□■

This forms the necessary conditions for a parallel decomposition. We now work on the sufficient conditions.

3.2.2 Sufficiency conditions for a parallel decomposition

The natural partitions corresponded to the components of a bi-parallel machine, and the determinism of the components forced them to be SP-partitions. The converse is also true—that is, given an SP partition the component, or quotient, machine "generated" by it is deterministic.

Definition 3.26

Given a (deterministic) machine, $\mathcal{M}$, and an SP partition of its states, π, we define the quotient machine *of $\mathcal{M}$ with π, denoted $\mathcal{M}/\pi$, to be the machine $\mathcal{M}_\pi$ where :-*

1. $Q_\pi = \pi$
2. $\Sigma_\pi = \Sigma$
3. $E_\pi = \{(U, a, V) \mid \Delta[U, a] \cap V \neq \emptyset\}$
4. $s_\pi = s|_\pi$
5. *If $\mathcal{M}$ is a finite machine then $F_\pi = F|_\pi$, and if $\mathcal{M}$ is an ω-machine then $\mathcal{F}_\pi = \mathcal{F}|_\pi$.*

The picture in Figure 3.11 with minor changes, is an example of a quotient machine. It is $\mathcal{M}/\pi_A$.

Because we only consider deterministic machines, we want to know when a quotient machine is deterministic. It turns out that, because of the requirement on the partition inducing it, that this is always the case.

Lemma 3.27

Any quotient machine is deterministic.

Proof:

Let the quotient machine be $\mathcal{M}/\pi$. Take an edge in $\mathcal{M}/\pi$. By definition, it is of the form (U, a, V) where $U, V \in \pi$ and $a \in \Sigma$, and $\Delta[U, a] \cap V \neq \emptyset$. Because π is an SP partition, we know $\Delta[\pi, \Sigma] \leq \pi$, so that for each $U \in \pi$ and $a \in \Sigma$ there is a $V' \in \pi$ so that $\Delta[U, a] \subseteq V'$. V' is unique given U and a because π is a partition (distinct members are disjoint), so that $V' = V$. Thus the edge is determined by its first two components. The edge being arbitrary completes the proof.

■□■

Thus $\mathcal{M}_P/\pi_A$ is deterministic. In fact it is isomorphic to $\mathcal{M}_A$, and is representative of the general case. Thus quotient machines, and hence SP partitions, provide a way of constructing components of a large machine.

Lemma 3.28 (Shi87)

If $\mathcal{M}$ has two non-trivial orthogonal SP partitions, then it has a parallel decomposition.

Proof:

Let the two partitions be π_1 and π_2, let $\mathcal{M}_1$ and $\mathcal{M}_2$ be as $\mathcal{M}/\pi_1$ and $\mathcal{M}/\pi_2$ respectively. Let $\mathcal{M}_{\|} = \mathcal{M}_1 \parallel \mathcal{M}_2$. We show $\mathcal{M} \trianglelefteq \mathcal{M}_{\|}$.

The states of $\mathcal{M}_{\|}$ are of the form (U_1, U_2) where $U_1 \in \pi_1, U_2 \in \pi_2$, and for each $q \in Q$ there is a unique $U \in \pi_1$ and $V \in \pi_2$ so that $q \in U$ and $q \in V$ (by virtue of π_1, π_2 being partitions). Thus the map $\phi : Q \to Q_{\|}$ defined by $\phi(q) = (\pi_1|_q, \pi_2|_q)$ is well defined.

Now the orthogonality of the partitions ensures that $\phi_1|_q \cap \phi_2|_q \leq 1$ for every $q \in Q$, so that ϕ is injective. In addition, for finite machines

$$\begin{aligned} \phi[F] &= (\pi_1|_F, \pi_2|_F) \\ &= (F_1, F_2) \\ &= F_{\|} \end{aligned}$$

whilst for ω-machines

$$\begin{aligned} \phi[\mathcal{F}] &= (\pi_1|_{\mathcal{F}}, \pi_2|_{\mathcal{F}}) \\ &= (\mathcal{F}_1, \mathcal{F}_2) \\ &= \mathcal{F}_{\|} \end{aligned}$$

It only remains to check that $q \xrightarrow{a} q'$ implies $\phi(q) \xrightarrow{a} \phi(q')$.

Let q, a, q' be defined so that $q \xrightarrow{a} q'$. Let $U_1 \in \pi_1, U_2 \in \pi_2$ be defined by $(U_1, U_2) = \phi(q)$, and let $V_1 \in \pi_1, V_2 \in \pi_2$ be defined by $(V_1, V_2) = \phi(q')$.

Because $q \xrightarrow{a} q'$, we have that $\Delta(q, a) = q'$. But $q \in U_1$ and $q' \in V_1$ we have that $\Delta[U_1, a] \cap V_1 \neq \emptyset$. Thus, $\Delta_1(U_1, a) = V_1$.

Similarly $\Delta_2(U_2, a) = V_2$. This, along with the fact that all the machines in question are deterministic (so that we can talk about their transition functions), gives

$$\begin{aligned}
\phi(q') &= (V_1, V_2) \\
&= (\Delta_1(U_1, a), \Delta_2(U_2, a)) \\
&= \Delta_{\|}((U_1, U_2), a) \qquad \textit{by Definition 3.1} \\
&= \Delta_{\|}(\phi(q), a)
\end{aligned}$$

so that $\phi(q) \xrightarrow{a} \phi(q')$

Thus $\phi : \mathcal{M} \leq \mathcal{M}_{\|}$. The non-triviality of the partitions ensure that $|Q_1|, |Q_2| < |Q|$, so that $\phi : \mathcal{M} \trianglelefteq \mathcal{M}_{\|}$ and $\mathcal{M}$ has an parallel decomposition.

■□■

The above result ensures that $\mathcal{M}_P \trianglelefteq \mathcal{M}_F/\pi_A \| \mathcal{M}_F/\pi_B$ and $\mathcal{M}_F \trianglelefteq \mathcal{M}_F/\pi_P \| \mathcal{M}_F/\pi_T$ because none of the partitions are trivial, and $\pi_A.\pi_B = \bot = \pi_P.\pi_T$. As mentioned earlier, they are in fact isomorphic, but, with allowance for inclusion, are representative.

A comment on the relationships involved

Although discussed earlier Robot A is not structurally contained in Pline or Factory, and is not homomorphic either, it turns out that both Pline and Factory *are* homomorphic to Robot A. This seems counter intuitive, because excepting the injective-ness of the map, homomorphic relationships seem to be structural inclusion, and it seems strange to describe a large machine made up from components as being "contained in" its components rather than the other way round. This highlights the necessity for the injectivity of any map which we want to represent structural inclusion.

As un-intuitive as this seems, in light of the language theoretic properties ascribed to the relationships of parallel compositions and homomorphisms—refer to Lemma 3.7 and Lemma 3.15—it is perhaps not so surprising. Also, the fact the each state in a component machine "corresponds" to many states in a parallel machine, suggests that a map in the other direction might be expected.

One last point in this note is that Pline and Factory are not representative in this instance. The existence of a *total* map depends upon the structural equivalence of both Pline and Factory with the parallel composition of their respective components. Structural inclusion only guarantees that a submachine (or, equivalently, a partial map), will have the desired property.

3.2.3 Bringing it together

The tools we have developed in the previous sections, now enable us to give us the answers we really want, but it is not immediate. First we note that all the results in Section 3.2.1, could be extended to a many component case very naturally (but with more notation) to give the following result (note that infinite collections are not a problem, as there are only a finite number of distinct partitions for the state set of a given machine).

Proposition 3.29

If a machine has a parallel decomposition then there is a collection of non-trivial SP partitions of its states whose product is trivial.

One more thing needs to be observed in order to clarify the final result. That is, for a given machine, the set of SP partitions of its states forms a lattice.

Lemma 3.30

The set of SP partitions of the states of a machine forms a complete lattice.

Proof:

Given a machine $\mathcal{M}$, the set of SP partitions on Q, is a subset of $\Pi(Q)$, itself a complete distributive lattice (Proposition 2.16). We use the same partial order, and note that the trivial partitions of Q are SP partitions, so that all that need to be shown is that for any two SP partitions, their product is also an SP partition.

Take two SP partitions, π_1 and π_2. By definition of an SP partition $\Delta[\pi_1, \Sigma] \leq \pi_1$ and $\Delta[\pi_2, \Sigma] \leq \pi_2$.

Take an arbitrary member of $\pi_1.\pi_2$. It is of the form $U_1 \cap U_2$ where $U_1 \in \pi_1$ and $U_2 \in \pi_2$. Now for each $a \in \Sigma$ there is a $V_1 \in \pi_1$ so that $\Delta[U_1, a] \subseteq V_1$ and a $V_2 \in \pi_2$ so that $\Delta[U_2, a] \subseteq V_2$. Now,

$$\begin{aligned} \Delta[U_1 \cap U_2, a] &\subseteq \Delta[U_1, a] \cap \Delta[U_2, a] \\ &\subseteq V_1 \cap V_2 \\ &\in \pi_1.\pi_2 \end{aligned}$$

The arbitrariness of the member of $\pi_1.\pi_2$ implies that $\Delta[\pi_1.\pi_2, a] \leq \pi_1.\pi_2$, so that $\pi_1.\pi_2$ is an SP partition as required.

■□■

This observation illuminates the desired result.

Theorem 3.31

A machine has a parallel decomposition iff it has two non-trivial orthogonal SP partitions of its states.

Proof:

If the machine in question has two non-trivial orthogonal SP partitions then, by Lemma 3.28, it has a parallel decomposition.

For the converse, by Proposition 3.29, if the machine has a parallel decomposition then there is a collection of non-trivial SP partitions whose product is trivial. Because the state set is finite, there can only be finitely many distinct SP partitions. Order them, take the first, and multiply by the second and then the third and so on, until the product is orthogonal. Suppose this requires the first k. By hypothesis, the product of the first $k-1$, π, is non-trivial. Also the k-th partition, ρ, is non-trivial. Because the set of SP partitions of the states of a machine form a lattice, π is an SP partition. π and ρ are the two non-trivial orthogonal SP partitions required.

■□■

3.2.4 Applying the results

So far we have seen in theory how one might go about breaking up a large untimed machine into parallel components, but the only examples have been machines we defined as parallel machines anyway. We now illustrate the use, by applying the earlier theory to an example that is not defined as a parallel composition.

Consider the machine in Figure 3.12—it is taken from [Zie87] and represents the behaviour of a two-buffer which can be reset, via a "d" action, whenever the numbers of "a" and "b" actions processed thus far are equal. The buffer must be cleared before processing a third "a" or "b".

Suppose we want to decompose this machine, $\mathcal{M}$, in the manner discussed earlier. One way of doing so is to find orthogonal non-trivial SP partitions. Finding these can be difficult, and here we will not be concentrating on these difficulties.

The non-triviality requirement will require that at least two states be grouped together. Pick two states arbitrarily (in practice we would order the states and work through them systematically), say 1 and 2. Group these together. We now try to fulfill the SP requirement, and find the minimal SP partition that groups these two states together.

Using the fact that the transition diagrams generated by SP partitions are deterministic (in fact, that

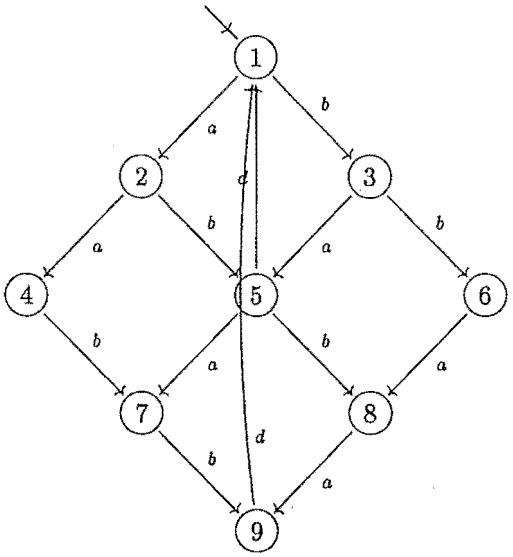

Figure 3.12: A "big" two-buffer

characterises them), we know that the group containing 1 and 2 must have a loop on an a transition, because $1 \xrightarrow{a} 2$ in $\mathcal{M}$. Because $2 \xrightarrow{a} 4$ in $\mathcal{M}$ also, determinacy requires that 4 also be grouped with 1 and 2. That is, Figure 3.13 is not part of a valid SP partition.

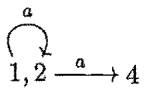

Figure 3.13: A non SP partition

Now because $1 \xrightarrow{b} 3$, $2 \xrightarrow{b} 5$, and $4 \xrightarrow{b} 7$, from the group containing 1, 2, and 4 there must be a b transition, and determinacy will force 3, 5, and 7 to be in the same group. Thus the groupings so far are shown in Figure 3.14.

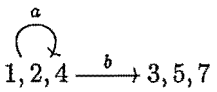

Figure 3.14: Building up an SP partition

Similarly 6, 8, and 9 must be in the same group, and this ends up being the only groups required by determinacy, as shown in Figure 3.15.

The minimal SP partition with 1 and 2 in the same group is thus non-trivial. We now try to find an SP partition orthogonal to this first one. Grouping 1 and 4 would not result in a partition orthogonal to the one in Figure 3.15, as their product would still group 1 and 4. Grouping 1 and 3, may do however, as 1

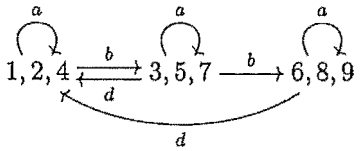

Figure 3.15: A minimal SP partition

and 3 are not grouped by the partition in Figure 3.15.

Looking at the minimal SP partition which groups 1 and 3 results in the picture in Figure 3.16.

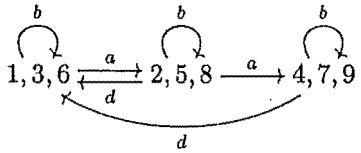

Figure 3.16: Another minimal SP partition

The two partitions thus generated are orthogonal, and thus $\mathcal{M}$ has a parallel decomposition. Note that although the state map is onto, there are transitions in the parallel composition of these two component machines which are not present in $\mathcal{M}$. For example, let the parallel composition machine be in state $(\{6,8,9\},\{2,5,8\})$, which corresponds to state 8 in $\mathcal{M}$. Because $\{6,8,9\} \xrightarrow{d} \{1,2,4\}$ in the first component and $\{2,5,8\} \xrightarrow{d} \{1,3,6\}$ in the second, this means that

$$(\{6,8,9\},\{2,5,8\}) \xrightarrow{d} (\{1,2,4\},\{1,3,6\})$$

in the parallel composition. This corresponds to a transition from state 8 to state 1 in $\mathcal{M}$, which is impossible. Thus $\mathcal{M}$ is strictly contained in, that is epimorphic to but not isomorphic to, some parallel composition.

We leave this section with a couple of notes and comments from the above example. It was fortuitous that the first two partitions we observed were orthogonal, and that neither of them forced everything to be grouped together, although for this particular example no transition formed as a "minimal" SP partition with a particular pair of states grouped together is trivial, often it is. These "minimal" SP partitions are often called prime partitions, and Theorem 3.31 could be strengthened to say that a machine has a parallel decomposition iff it has one with two orthogonal prime SP partitions.

Also using the machine in Figure 3.12 as an example, it is possible to get a parallel decomposition where the state set of the parallel machine is strictly greater than that of $\mathcal{M}$. Simply replace the second partition

with the minimal SP partition grouping 1 and 5. This new partition is

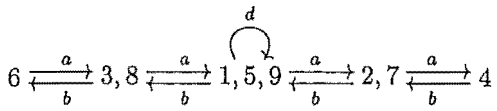

Figure 3.17: Another minimal SP partition

The partitions in Figure 3.15 and Figure 3.17 are orthogonal, non-trivial, and the quotient machines constructed from them have state sets of size 3 and 5 respectively. Thus their composition, which $\mathcal{M}$ will be epimorphic to, will have 15 states, whereas $\mathcal{M}$ only has 9. Thus the state map will no longer be onto.

Thus, the example developed throughout this section both shows that it is possible to have a machine strictly contained in its parallel decomposition, and illustrates a potential method for utilising Theorem 3.31 to effect a parallel decomposition. Informally, this method involves looking at minimal SP partitions, which are built up by considering the smallest (with respect to $\leq$ as defined in Definition 2.5) SP partitions which at least group a pair of elements. Implementation is not the focus of this work, as we concentrating on the extensions of this theory to other sorts of machines. Suffice it to mention that there is a decidable, albeit unwieldy, algorithm for finding orthogonal SP partitions [Shi87].

3.2.5 Summary

In this chapter we have reviewed the existing parallel decomposition theory for finite machines, and simultaneously presented an equivalent collection of results for ω-machines—the reason for the concurrent presentation is that the structural presentation of both sorts of machines is similar enough that many of the results are word-for-word identical.

The culmination of this chapter was a necessary and sufficient condition for the existence of a parallel decomposition of a finite or ω-machine. This condition was given in terms of a particular sort of partition, which provides a mechanism for finding parallel decompositions if they exist, and if they don't exist then showing that. In the next chapter we try to use many of the same concepts to develop a similar result for timed machines. Their structural presentation is dissimilar enough that the results require more effort than was the case for ω-machines.

Chapter 4

Decomposition of Timed Machines

This chapter provides some extensions to the existing parallel decomposition theory from finite machines to timed machines. The results do not flow over as directly as they did for the ω-machine case because we need to account for the clocks. We attempt to proceed in much the same fashion for as long as possible. Thus we keep in mind the same intuitions that motivated the earlier notions of composition, homomorphism and decomposition in Chapter 3, and even use similar examples for motivation.

4.1 Composition of Timed Machines

The formal description of a timed machine was presented in Definition 2.32. We now present a formal description of a timed machine that will represent the parallel composition of timed machines. The intuitive idea behind a parallel composition is that each machine will process the inputs at the same rate, the global machine only being able to process each input if each of its component machines can also. For the present we assume that the set of all actions/inputs is the same for all the machines in question. This is part of a more general theory which has each machine only processing a subset of the inputs, and common actions cause component synchronisation. Here, because all the alphabets are the same, each component processes all the inputs, thus giving a lockstep composition.

If we consider, for the sake of simplicity, just the case where there are two component machines, say $\mathcal{M}_1$ and $\mathcal{M}_2$. These combine together to form the parallel machine, $\mathcal{M}_{\|}$. The states of $\mathcal{M}_{\|}$ will be ordered pairs, where the first component will be the state of $\mathcal{M}_1$ and the second will be the state of $\mathcal{M}_2$. Since $\mathcal{M}_1$ and $\mathcal{M}_2$ may each be in any of their states, $Q_{\|}$ will be equal to (Q_1, Q_2) (recalling the convention that subscripts of machines carry onto their components). Similarly, the global machine starts with each

of its components in their starting positions, so that $s_{\|} = (s_1, s_2)$.

With the same intuition in mind we see that the stopwatches of $\mathcal{M}_{\|}$ are those of both $\mathcal{M}_1$ and $\mathcal{M}_2$, so that $C_{\|} = C_1 \uplus C_2$. Note that this is not the union of the two sets as there may inadvertently be stopwatches with the name in both $\mathcal{M}_1$ and $\mathcal{M}_2$.

In a similar vein to the developed example of Chapter 3 we consider a simplified factory. In order to highlight the similarities and differences between the untimed theory and the time one we have only two parts to our factory, but each part is more complicated than before.

Consider the situation depicted in Figure 4.1 describing the flow of a product through a factory.

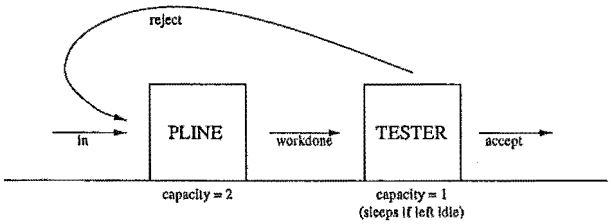

Figure 4.1: A simple factory

The system consists of two part, one part, ROBOT, receives raw materials and processes them, producing a product which is then passed to the other part, TESTER, which examines the product and either accepts it, in which case it passes out of the system, or rejects it, in which case the product gets returned to ROBOT as raw materials.

The intuitive meanings of the labels on the arrows are thus as follows. The "*in*" arrow corresponds to the passing of the raw materials to ROBOT; the "*workdone*" arrow corresponds to the passing of the product from ROBOT to TESTER; the "*accept*" and "*reject*" arrows correspond to the TESTER accepting or rejecting the product respectively.

Suppose we have the following conditions on the behaviour of the system; ROBOT has a workbench capacity of two, but if a second batch of raw materials arrives whilst it is processing the first then it must complete work on both batches before receiving any more. It also takes less than two time units from the time a batch arrives until ROBOT has completed working on that batch. TESTER only has a workbench of one, but has a turnaround time of no more than one time unit. If nothing occurs for more than four time units TESTER goes to sleep to preserve power. One description of the above is illustrated by the timed machines depicted in Figure 4.2, where we are once more observing the convention that transitions on actions that never change the state of a particular machine are not shown for that machine.

We want to describe the behaviour of the system as a whole. We do this in the manner detailed below.

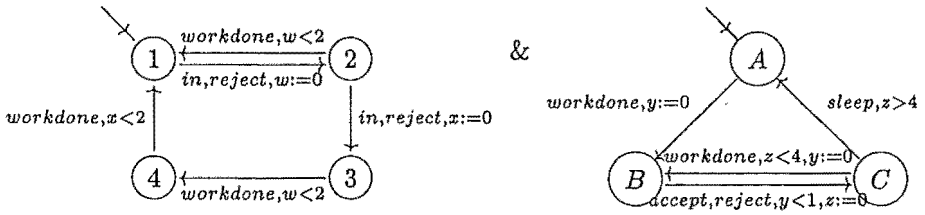

Figure 4.2: ROBOT and TESTER as separate timed machines, all cycles accepted

The edges of the global machine will each correspond to a pair of edges, one in each component—with the constraints on the edge such that the edge can be traversed precisely when each the corresponding edges in each component can be traversed.

The only arbitrary part of the definition is the set of final states, and to keep with the Cartesian product nature of the construction we define it here to be $(\mathcal{F}_1, \mathcal{F}_2)$. Combining all of these gives the following definition.

Definition 4.1

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two timed automata with the same alphabet. We define their parallel composition, *$\mathcal{M}_1 \| \mathcal{M}_2$ to be the machine $\mathcal{M}_\|$, where :*

1. $Q_\| = (Q_1, Q_2)$,
2. $\Sigma_\| = \Sigma_1 = \Sigma_2$,
3. $s_\| = (s_1, s_2)$,
4. $C_\| = C_1 \uplus C_2$,
5. $E_\| \subseteq (Q_\|, Q_\|, \Sigma_\|, 2^{C_\|}, \Phi(C_\|))$ *is equal to*
   $\{e_\| \mid \exists e_1 \in E_1, e_1 = (q_1, r_1, a, X_1, \delta_1) \in E_1, \exists e_2 \in E_2, e_2 = (q_2, r_2, a, X_2, \delta_2) \in E_2,$ *where*
   $e = ((q_1, q_2), (r_1, r_2), a, X_1 \uplus X_2, \delta_1 \wedge \delta_2)\}$
6. $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$

So for the example from Figure 4.1, if we construct the parallel composition of ROBOT and TESTER, we generate the machine shown in Figure 4.3—note that, in order to keep the transition structure transparent only some of the transitions are labelled, and even these are abbreviated. The labels on each unlabelled transition, however, are identical to that of a labelled transition which it is parallel to. The labels are as follows.

- α stands for "$workdone, x < 2 \wedge z < 4, y := 0$".
- β stands for "$workdone, w < 2 \wedge z < 4, y := 0$".
- γ stands for "$reject, y < 1, z := 0$".

In addition the state xy corresponds to the state (x, y).

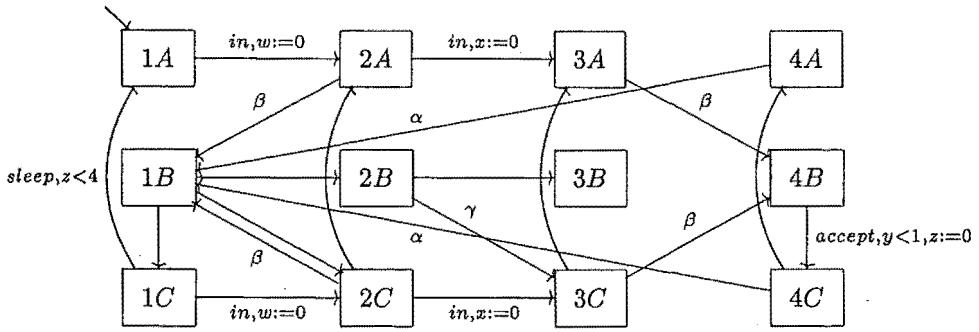

Figure 4.3: FACTORY—the parallel composition of ROBOT and TESTER

Just as was the case in Chapter 3, the definition for two components, Definition 4.1, has a natural extension to many components.

Definition 4.2

Let $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_n$ be a collection of timed machines—we define their parallel composition *to be the machine $\mathcal{M}_{\|}$ where :*

1. $Q_{\|} = (Q_1, Q_2, ..., Q_n)$
2. $\Sigma_{\|} = \Sigma_1 (= \Sigma_2 = ... = \Sigma_n)$
3. $s_{\|} = (s_1, s_2, ..., s_n)$
4. $C_{\|} = C_1 \uplus C_2 \uplus ... \uplus C_n$
5. $E_{\|} \subseteq (Q_{\|}, \Sigma, Q_{\|})$ *is defined as*

$$E_{\|} = \{(q_{\|}, r_{\|}, a, X_{\|}, \delta_{\|}) \mid q_{\|} = (q_1, q_2, ..., q_n), r_{\|} = (r_1, r_2, ..., r_n)$$
$$X_{\|} = X_1 \uplus X_2 \uplus ... \uplus X_n, \delta_{\|} = \delta_1 \wedge \delta_2 ... \delta_n \text{ where for each } i \in \{1, 2, ..., n\}$$
$$(q_i, r_i, a, X_i, \delta_i) \in E_i\}$$

6. $F_{\|} = (F_1, F_2, ..., F_n)$

The remainder of this section is devoted to the results relating the projections and combinations of automata and their runs.

The motivation of a parallel composition being of independent components where the global machine can only move if each of the components can leads one to suspect that if each component is deterministic (that is, there is only one transition for each action and extended state) then the global machine will also be deterministic. This is the case for timed machines as it was for both finite and ω-machines (Lemma 3.5). The proof is also similar. In fact the presentation of the material in Chapter 3 was presented, deliberately, in a manner that will ensure that all of the results of this chapter are natural extensions of the results there.

Lemma 4.3

If $\mathcal{M}_{\|}$ is the parallel composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ then $\mathcal{M}_{\|}$ is deterministic iff both $\mathcal{M}_1$ and $\mathcal{M}_2$ are.

Proof:

Formally,

$$
\begin{array}{lll}
 & \mathcal{M}_{\|} \textit{ is deterministic} & \\
\Leftrightarrow & \forall q_{\|} \in Q_{\|}, \forall \nu_{\|} \in (\mathbb{R}^+)^{C_{\|}}, \forall a \in \Sigma & \textit{Definition } 2.32 \\
 & \exists! e_{\|} \in E_{\|},\ e_{\|} = (q_{\|}, r_{\|}, a, \delta_{\|}, X_{\|}) \in E_{\|} \textit{ and } (\nu_{\|} \models \delta_{\|}) & \textit{Definition } 4.2 \\
\Leftrightarrow & \forall (q_1, q_2) \in (Q_1, Q_2), \forall \nu \in (\mathbb{R}^+)^{C_1 \uplus C_2}, \forall a \in \Sigma & \\
 & \exists! (e_1, e_2) \in (E_1, E_2) \textit{ such that} & \\
 & e_i = (q_i, r_i, a, \delta_i, X_i) \textit{ for } i = \{1,2\}, \textit{ and } (\nu \models \delta_1 \wedge \delta_2) & \textit{Figure } 2.1.4 \\
\Leftrightarrow & \forall (q_1, q_2) \in (Q_1, Q_2), \forall (\nu_1, \nu_2) \in ((\mathbb{R}^+)^{C_1}, (\mathbb{R}^+)^{C_2}), \forall a \in \Sigma & \\
 & \exists! (e_1, e_2) \in (E_1, E_2) \textit{ such that} & \\
 & e_i = (q_i, r_i, a, \delta_i, X_i), \textit{ and } (\nu_i \models \delta_i) \textit{ for } i = \{1,2\} & \textit{Definition } 4.2 \\
\Leftrightarrow & \forall i \in \{1,2\}, \forall (q_i, \nu_i, a) \in (Q_i, (\mathbb{R}^+)^{C_i}, \Sigma) & \\
 & \exists! e_i \in E_i \textit{ such that } e_i = (q_i, r_i, a, \delta_i, X_i) \textit{ and } (\nu_i \models \delta_i) & \\
\Leftrightarrow & \mathcal{M}_1 \textit{ and } \mathcal{M}_2 \textit{ are deterministic} &
\end{array}
$$

□

Informally, the definition of determinism states that there is only one edge that can be traversed for each extended state and action. Because of the definition of an edge in a parallel machine we know that for each extended state and action there is only one edge in each component such that their combination can be traversed by the parallel machine in that extended state.

Because the clock sets in the components are disjoint, the conjunction of a constraint from each edge is satisfied precisely when the restriction of the constraint to each clock set is satisfied by the projection of the time state onto that clock set. Also each time state for the parallel machine corresponds to a unique pair of time states—one from each component.

This means that for each component there is a unique edge that can be traversed for each extended state and action. That is, each component is deterministic. This argument can be reversed to provide the converse.

■□■

This theorem guarantees that, because both ROBOT and TESTER are deterministic, their parallel composition will also be deterministic. FACTORY is indeed deterministic, verifying the theorem.

Similarly, the notion of a parallel composition being a lockstep one means that a transition in the global machine will correspond to a transition in each of its components, as demonstrated by the following lemma.

Lemma 4.4

If $\mathcal{M}_{\|} = \mathcal{M}_1 \| \mathcal{M}_2$ *then*

$$((q_1, q_2), \nu_1 \uplus \nu_2) \xrightarrow{a,t} ((r_1, r_2), \xi_1 \uplus \xi_2)$$

in $\mathcal{M}_{\|}$ *iff for each of* $i = 1, 2$, $\quad (q_i, \nu_i) \xrightarrow{a,t} (r_i, \xi_i) \quad$ *in* $\mathcal{M}_i$.

Proof:

From Lemma 4.3 we know that $f_{\|}$ is defined iff both f_1 and f_2 are defined. Thus for every

$$((q_1, q_2), (\nu_1, \nu_2), a, t, (r_1, r_2), (\xi_1, \xi_2)) \in (Q_{\|}, (0, \infty)^{C_{\|}}, \Sigma, \mathbb{R}^+, Q_{\|}, (0, \infty)^{C_{\|}})$$

we have the following.

$$\begin{aligned}
& ((q_1, q_2), \nu_1 \uplus \nu_2) \xrightarrow{a,t} ((r_1, r_2), \xi_1 \uplus \xi_2) \\
\Leftrightarrow \quad & f_{\|}((q_1, q_2), \nu_1 \uplus \nu_2 + t, a) = ((r_1, r_2), X_1 \cup X_2, \delta_1 \wedge \delta_2) \\
& \textit{where } \nu_1 \uplus \nu_2 + t \models \delta_1 \wedge \delta_2, \textit{ and } (\xi_1, \xi_2) = [X_1 \cup X_2 \to 0](\nu_1 \uplus \nu_2 + t) \quad \textit{Definition 2.32} \\
\Leftrightarrow \quad & \forall i \in \{1, 2\}, f_i(q_i, \nu_i + t, a) = (r_i, X_i, \delta_i) \\
& \textit{where } \nu_i + t \models \delta, \textit{ and } \xi_i = [X_i \to 0](\nu_i + t) \quad \textit{Figure 2.1.4} \\
\Leftrightarrow \quad & \forall i \in \{1, 2\} \; (q_i, \nu_i) \xrightarrow{a,t} (r_i, \xi_i) \quad \textit{Definition 2.32}
\end{aligned}$$

as required.

□

The intuition behind the proof proceeds as follows. If we assume the LHS then from the definition of a transition there must be an associated edge, whose constraint is satisfied by the time state. By the definition of edges in a $\mathcal{M}_{\|}$, this corresponds to an edge in each component. Restricting the time state to each clock set means that the corresponding transitions are also possible in the component machines. Once more, the argument may be reversed to provide the converse.

■□■

To provide an illustration of this, we continue to develop the ROBOT-TESTER-FACTORY example. We denote the timed machines representing the behaviour of ROBOT, TESTER, and FACTORY as $\mathcal{M}_T$, $\mathcal{M}_R$ and $\mathcal{M}_F$ respectively. Thus, because

$$(2,0) \xrightarrow{workdone,0} (1,0)$$

is a valid transition in $\mathcal{M}_R$ and

$$(A,0) \xrightarrow{workdone,0} (B,0)$$

is a valid transition in $\mathcal{M}_T$ then it must be the case that

$$((2,A),0) \xrightarrow{workdone,0} ((1,B),0)$$

is a valid transition in $\mathcal{M}_F$. Similarly, because

$$(2,\{(w,5),(x,5)\}) \xrightarrow{reject,0.5} (3,\{(w,5.5),(x,0)\})$$

is a valid transition in $\mathcal{M}_R$ and

$$(C,\{(y,0.3),(z,10)\}) \xrightarrow{reject,0.5} (B,\{(y,0.8),(z,0)\})$$

is a valid transition in $\mathcal{M}_T$ then it must be the case that

$$((2,C),\{(w,5),(x,5),(y,0.3),(z,10)\}) \xrightarrow{reject,0.5} ((3,B),\{(w,5.5),(x,0),(y,0.8),(z,0)\})$$

is a valid transition in $\mathcal{M}_F$. We can observe that both the above are indeed true by looking at Figure 4.3.

The only arbitrary part in the definition of the parallel composition was the definition of the final sets. Our intuitive notion of a lockstep parallel composition requires merely that transitions for the global machine are those for both component machines. It tells us nothing about any acceptance criteria. The acceptance criteria that we have included in Definition 4.2 is perhaps the most obvious and gives the

global machine the property that any word accepted by it is a word accepted by its components and vice versa, as formalised below. It is also in keeping with the spirit of parallel composition as Cartesian product.

Lemma 4.5

If $\mathcal{M}_{\|}$ is the parallel composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ then $L(\mathcal{M}_{\|}) = L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$

Proof:

$(\alpha, \tau) \in L(\mathcal{M}_{\|})$

$\Leftrightarrow$ *there is a successful run,* $(\sigma, \Upsilon) \in \left((Q_{\|}, (0,\infty)^{C_{\|}})\right)^{\omega}$, *of* (α, τ) *on* $\mathcal{M}_{\|}$. *Definition* 2.32

$\Leftrightarrow$ $\exists (\sigma, \Upsilon) \in (Q_{\|}^{\omega}, ((R^+)^{C_{\|}})^{\omega})$

$\sigma(0) = 0, \Upsilon(0) = 0, In(\sigma) \in \mathcal{F}_{\|}$,

$\forall i \in \omega,\ (\sigma(i), \Upsilon(i)) \xrightarrow{\alpha(i), \Delta\tau(i)} (\sigma(i+1), \Upsilon(i+1))$

Let $\sigma(i) = (\sigma_1(i), \sigma_2(i)), \Upsilon(i)|_{C_1} = \Upsilon_1(i)$, *and* $\Upsilon(i)|_{C_2} = \Upsilon_2(i)$.

$\Leftrightarrow$ $\forall i \in \{1,2\}, \exists (\sigma_i, \Upsilon_i) \in (Q_i^{\omega}, ((R^+)^{C_i})^{\omega})$

$\sigma_i(0) = 0, \Upsilon_i(0) = 0, In(\sigma_i) \in \mathcal{F}_i$,

$\forall j \in \omega,\ (\sigma_i(j), \Upsilon_i(j)) \xrightarrow{\alpha(j), \Delta\tau(j)} (\sigma_i(j+1), \Upsilon_i(j+1))$ *Lemma* 4.4

$\Leftrightarrow$ $\forall i \in \{1,2\}$, *there is a successful run of* (α, τ) *on* $\mathcal{M}_i$.

$\Leftrightarrow$ $(\alpha, \tau) \in L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$

□

Less formally, using Lemma 4.4 we know that a run, $(\sigma_{\|}, \Upsilon_{\|}($, on $\mathcal{M}_{\|}$ corresponds to a run in each component, (σ_i, Υ_i) for $i = 1, 2$. Also the infinite set of $\sigma_{\|}$ will consist of all the ordered pairs where the ith components are members of the infinite set of $\sigma_{\|}$—by virtue of the finiteness of all the state sets.

The definition of $\mathcal{F}_{\|}$ thus ensures that successful runs on $\mathcal{M}_{\|}$ correspond precisely to successful runs on both $\mathcal{M}_1$ and $\mathcal{M}_2$ giving the desired result.

■□■

Describing the language of a machine directly was sufficiently complex for untimed machines (Lemma 3.7), and it is more so for timed machines. As a result, we will only illustrate this example by looking at sequences rather than languages. Thus, for example, the timed word α given by

$$(in, 0.5), (workdone, 2), (accept, 5), (in, 10.5), (workdone, 12), ...$$

is accepted by ROBOT, that is, the sequence of extended states

$$(1,0),(2,\{(w,0),(x,0.5)\}),(1,\{(w,1.5),(x,2)\}),(1,\{(w,4.5),(x,5)\}),...$$

is an accepting run of α on $\mathcal{M}_R$. Thus $\alpha \in L(\mathcal{M}_R)$. However there is no accepting run of α on $\mathcal{M}_T$, so that $\alpha \notin L(\mathcal{M}_T)$. The above result thus ensures that $\alpha \notin L(\mathcal{M}_F)$ which is indeed the case.

On the other hand, the timed word β given by

$$(in,1),(workdone,2.5),(reject,3),(workdone,4.5),(reject,5),...$$

is accepted by both ROBOT and TESTER. The above theorem thus guarantees that $\beta \in L(\mathcal{M}_F)$ which is indeed the case.

4.2 Comparison of Timed Machines

Once more we will define our homomorphisms as maps that correspond to some form of structural inclusion. Because the formal description of a timed machine has clocks as well as states, both these will be mapped. Although the states map forward, because the time states are themselves maps from the clock set, in order to get the extended states to correspond the time states must be composed with the inverse of the clock maps.

For example, we wish to consider the machines depicted in Figure 4.4 to be structurally contained in one another. The "additional" state in the machine on the right is unreachable, and thus has no effect on the behaviour of the machine, but equivalently the transition could be from Q to T rather than from T to Q, and the structural inclusion will still hold. The requirement is not that additional states and transitions be unreachable, merely that there there will be no fewer states or transitions in a "bigger" machine.

Thus, we require maps for both the states and the clocks. A homomorphism doesn't require that they be injective, but our concept of structural inclusion will. The starting states must correspond, as must the accepting sets.

Now, suppose the "smaller" machine is in a particular extended state, say (q,ν) and can do a transition on a and t to move to another extended state, (r,ξ). We want the larger machine to be able to perform the "same" transition. What we mean by the "same" transition is that there will be an associated extended states in the larger machine such that the extended state associated with (q,ν) can do a transition to the extended state associated with (r,ξ). Thus we need to introduce a way of associating extended states of the smaller machine with those of the larger machine.

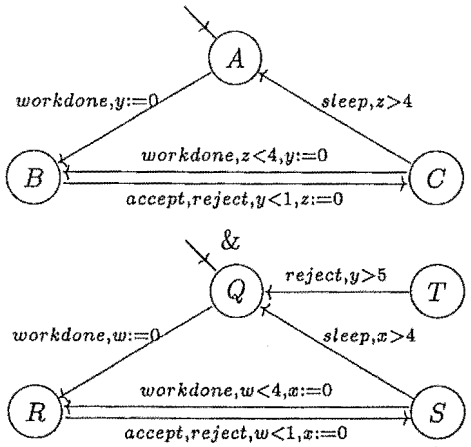

Figure 4.4: TESTER and timed machine which is structurally larger than TESTER

The choice of which state in the larger machine to associate with a given state in the smaller machine is very natural . It is the same as with untimed machines. If the state in the smaller machine is q and the state map is ϕ then the state in the larger machine that corresponds to q is its image under ϕ, that is $\phi(q)$ is associated with q. The time states are not as easy, because each time state is itself a function. If the clock map is η and the two time states are ν in the smaller machine and ν^* (which we desire to find) in the larger machine, then we have Figure 4.5.

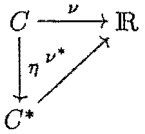

Figure 4.5: Time state correspondences.

The natural correspondence sees Figure 4.5 commute, that is choose ν^* so that $\nu^* \circ \eta = \nu$. This means that $\nu^* = \nu \circ \eta^{-1}$ over any domain where η^{-1} is a function. When η^{-1} is not a function then, in order to keep the concept of inclusion ν^* will have to satisfy any clock constraints that ν could. To enable this we let $\nu \circ \eta^{-1}$ only be defined over the largest domain where η^{-1} is a function. That is, for all other points, x, $\nu \circ \eta^{-1}(x) = *$. We now allow that a $*$ clock value will cause any clock constraint using that clock to be satisfied. This will not matter for the majority of the chapter as structural inclusion will require that η be injective. Thus we have the following.

Definition 4.6

Given two timed machines, $\mathcal{M}$ and $\mathcal{M}_*$, a homomorphism *between $\mathcal{M}$ and $\mathcal{M}_*$ is an ordered pair (ϕ, η) where $\phi : Q \to Q_*$ and $\eta : C \to C_*$ are such that :*

1. $\phi(s) = s_*$

2. *If* $(q,\nu) \xrightarrow{a,t} (r,\xi)$ *then* $(\phi(q),\nu\circ\eta^{-1}) \xrightarrow{a,t} (\phi(r),\xi\circ\eta^{-1})$

3. $\phi[\mathcal{F}] \subseteq \mathcal{F}_*$.

We write $(\phi,\eta) : \mathcal{M} \rightsquigarrow \mathcal{M}_*$ *if the above holds. If* ϕ *and* η *are both onto then we say we have an* epimorphism. *If they are both total, then we say we have a* total *homomorphism, and if furthermore, they are both 1-1 then we say we have a* monomorphism. *A monomorphism whose inverse (which necessarily exists) is also a monomorphism is called an* isomorphism.

If $\mathcal{M}$ *and* $\mathcal{M}_*$ *have a homomorphism between them then we say that they are* homomorphic *to one another. Similarly machines may be totally homomorphic, epimorphic, monomorphic, or isomorphic.*

The first condition ensures that the initial states correspond, the second that transitions in the image machine at least match those in the original, and the third that the original machine has stricter acceptance criteria.

The machine on the left of Figure 4.4 (TESTER) is monomorphic to the machine on the right, with state map, ϕ defined as

$$\phi = \{(A,Q),(B,R),(C,S)\}$$

and clock map η defined as

$$\eta = \{(y,w),(z,x)\}$$

Once more the language relationship between the sets of accepted words reflects the motivation for the definition of a homomorphism.

Lemma 4.7

If $\mathcal{M}$ *is homomorphic to* $\mathcal{M}_*$ *then the behaviour of* $\mathcal{M}$ *is contained in the behaviour of* $\mathcal{M}_*$.

Proof:

We show the result by relating a run of $\mathcal{M}$ on a particular word to a run of $\mathcal{M}_*$ on the same word.

Let $(\phi,\eta) : \mathcal{M} \rightsquigarrow \mathcal{M}_*$ and suppose $(\alpha,\tau) \in L(\mathcal{M})$. Then there is a successful run, (σ,Υ), of (α,τ) on $\mathcal{M}$. Now $\sigma(0) = 0, \Upsilon(0) = 0, In(\sigma) \in \mathcal{F}$ and for each $i \in \omega$ we have from Definition 2.32 that

$$(\sigma(i),\Upsilon(i)) \xrightarrow{\alpha(i),\Delta\tau(i)} (\sigma(i+1),\Upsilon(i+1))$$

Let $(\sigma_*, \Upsilon_*) \in (Q_*^\omega, (0,\infty)^\omega)$ be defined by $\sigma_* = \phi[\sigma], \Upsilon_*(i) = \Upsilon(i) \circ \eta^{-1}$.

Now $\sigma_*(0) = 0, \Upsilon_*(0) = 0$, and $In(\sigma_*) \in \phi[\mathcal{F}] \subseteq \mathcal{F}_*$. Also, from Definition 4.6, we know for every $i \in \omega$ that $(\sigma_*(i), \Upsilon_*(i)) \xrightarrow{\alpha(i), \Delta\tau} (\sigma_*(i+1), \Upsilon_*(i+1))$, so that (σ_*, Υ_*) is a successful run of (α, τ) on $\mathcal{M}_*$. Thus $(\alpha, \tau) \in L(\mathcal{M}_*)$, again using Definition 2.32.

Because (α, τ) was an arbitrary member of $L(\mathcal{M})$ we have that $L(\mathcal{M}) \subseteq L(\mathcal{M}_*)$ as required.

■□■

Thus, any word accepted by TESTER will be accepted by the machine on the left of Figure 4.4.

This result has the following corollary which further justifies the idea that an isomorphism is a structure independent relabelling.

Corollary 4.8

If $\mathcal{M}$ is isomorphic to $\mathcal{M}_*$ then $L(\mathcal{M}) = L(\mathcal{M}_*)$.

Before going on to discuss precisely when a timed machine has a parallel decomposition we first note the following property of homomorphisms and parallel compositions.

Lemma 4.9

If $\mathcal{M}$ is X-morphic to both $\mathcal{M}_1$ and $\mathcal{M}_2$ then it is X-morphic to $\mathcal{M}_1 \| \mathcal{M}_2$, where X- is one of homo-, total homo-, or mono-.

Proof:

Let $\mathcal{M}_\| = \mathcal{M}_1 \| \mathcal{M}_2$. For each $i \in \{1,2\}$, let $(\phi_i, \eta_i) : \mathcal{M} \rightsquigarrow \mathcal{M}_1$, and define

$$(\phi_\|, \eta_\|) = ((\phi_1 \| \phi_2), (\eta_1 \uplus \eta_2)))$$

Then,

1. $\phi_\|(s) = (s_1, s_2) = s_\|$, from Definition 4.2.
2. If $(q, \nu) \xrightarrow{a,t} (r, \xi)$ then, by Definition 4.6, for each $i \in \{1,2\}$,

$$(\phi_i(q), \nu \circ \eta_i^{-1}) \xrightarrow{a,t} (\phi_i(r), \xi \circ \eta_i^{-1})$$

By Lemma 4.4 $(q_*, \nu_*) \xrightarrow{a,t} (r*, \xi_*)$ where $q_* = (\phi_1(q), \phi_2(q))$, $\nu_* = \nu_1 \circ \eta^{-1} \uplus \nu_2 \circ \eta^{-1}$,

$r_* = (\phi_1(r), \phi_2(r))$ and $\xi_* = \xi \circ \eta_1^{-1} \uplus \nu \circ \eta_2^{-1}$ so that $(\phi_{\|}(q), \nu \circ \eta_{\|}^{-1}) \xrightarrow{a,t} (\phi_{\|}(r), \xi \circ \eta_{\|}^{-1})$ from Definition 4.2

3. $\phi_{\|}[\mathcal{F}] = (\phi_1(\mathcal{F}), \phi_2(\mathcal{F})) \subseteq (\mathcal{F}_1, \mathcal{F}_2) = \mathcal{F}_*$ from Definition 4.2 and Definition 4.6.

Together the above imply that $\mathcal{M}$ is homomorphic to $\mathcal{M}_1 \| \mathcal{M}_2$ as required for X- being homo-. For X- being total homo- note that that ϕ_i, η_i being total for each $i = 1, 2$ implies that $\phi_{\|}, \eta_{\|}$ are both total. Similarly for X- being mono- note that ϕ_i, η_i being injective for each $i = 1, 2$ implies that $\phi_{\|}, \eta_{\|}$ are both injective, completing the proof.

□

Informally, from the component maps we define $\phi_{\|}$ and $\eta_{\|}$ so that for each $q \in Q$ we have that

$$\phi_{\|}(q) = (\phi_1(q), \phi_2(q))$$

and for each $c \in C$ we have that

$$\eta_{\|}(c) = \begin{cases} \eta_1(c) & , c \in C_1 \\ \eta_2(c) & , c \in C_2 \end{cases}$$

We claim that $(\phi_{\|}, \eta_{\|})$ is the required X-morphism.

Checking the definitions of both homomorphism and parallel composition we see that the starting states correspond satisfying the first condition for $(\phi_{\|}, \eta_{\|})$ to be a homomorphism. Likewise the accepting sets fulfil the third condition. For the second condition take an arbitrary transition in $\mathcal{M}$. Because (ϕ_1, η_1) and (ϕ_2, η_2) are both homomorphisms, there is a corresponding transition in both $\mathcal{M}_1$ and $\mathcal{M}_2$. By Lemma 4.4, these two transitions correspond to a transition in $\mathcal{M}_{\|}$, and this transition is the one required to satisfy the second, and final, condition for $(\phi_{\|}, \eta_{\|})$ to be a homomorphism. Thus the result is true for X- being homo-. The totality/injectiveness of the component maps carry over immediately providing the remaining cases.

■□■

4.3 Decomposition of Timed Machines

The main thrust of this chapter is to find out what properties of a machine are sufficient to ensure that it is "contained in" a parallel composition of simpler machines. Our formal description of "containment"

is the existence of a monomorphism between the two machines, and a simpler machine will be one with fewer states. Thus we have the following definition.

Definition 4.10

A timed machine, $\mathcal{M}$, has a parallel decomposition *if there exist two timed machines, $\mathcal{M}_1$ and $\mathcal{M}_2$, such that $|Q_1|, |Q_2| < |Q|$ and there is a monomorphism $(\phi, \eta) : \mathcal{M} \rightsquigarrow \mathcal{M}_1 || \mathcal{M}_2$.*

Here is a picture showing the situation. The wavy arrow represents a monomorphism and the line-double arrow represents the part-whole relationship of a parallel composition. The dotted arrows describe the relationships that we will investigate to determine the requirements on $\mathcal{M}$ in order for it to have a parallel decomposition.

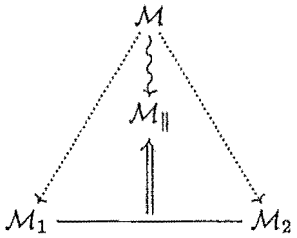

Figure 4.6: The relationship between $\mathcal{M}$ and $\mathcal{M}_{||}$.

We want to determine if a particular timed machine has a parallel decomposition or not. Initially, however, we will focus on some necessary conditions and will assume that the machine in question, $\mathcal{M}$, does have a parallel decomposition with the two component machines being $\mathcal{M}_1$ and $\mathcal{M}_2$, and monomorphism $(\phi, \eta) : \mathcal{M} \rightsquigarrow \mathcal{M}_1 || \mathcal{M}_2 = \mathcal{M}_{||}$.

To motivate the results we use the example introduced earlier in the chapter, where FACTORY was defined as the parallel composition of ROBOT and TESTER. With allowance for inclusion, as in Chapter 3, we will see that it is representative. Look at Figure 4.3. As in Chapter 3 notice that the states of FACTORY that correspond to ROBOT being in state 1, that is states $(1, A), (1, B), (1, C)$, are very naturally grouped together. Similarly, the other states of ROBOT generate natural groupings, and together they form a natural partition. Likewise, TESTER generates a natural partition of the states as well.

First we note that associated with each of $\mathcal{M}_1$ and $\mathcal{M}_2$, WLOG $\mathcal{M}_1$, there corresponds a natural partition, π_1, of the states of $\mathcal{M}$. That is, for each state, q_1, of $\mathcal{M}_1$, there will correspond a member of π_1 that contains all the states of $\mathcal{M}$ that map (under ϕ) to states of $\mathcal{M}_{||}$ where the first component is in state q_1. This follows the example of Definition 3.20 in that all information, be it state or clock information, in the parallel machine that comes from the second component is ignored.

Definition 4.11

If $\mathcal{M}$ has a parallel decomposition and $(\phi, \eta) : \mathcal{M} \rightsquigarrow \mathcal{M}_{\|}$ then the induced state partitions *of $\mathcal{M}$ are given by*

1. $\pi_1 = \{\phi^{-1}[(q_1, Q_2)] | q_1 \in Q_1\}$, *and*
2. $\pi_2 = \{\phi^{-1}[(Q_1, q_2)] | q_2 \in Q_2\}$.

Similarly, there is a natural partition of the clocks of $\mathcal{M}_{\|}$ corresponding to the clocks that are mapped to into each component.

Definition 4.12

The induced clock partition *of $\mathcal{M}$ is given by*

$$\{\eta^{-1}[C_1], \eta^{-1}[C_2]\}$$

Note that the appellation of partition is justified by η being total and $C_{\|}$ being $C_1 \uplus C_2$.

We will continue to use this notation as long as we are discussing the necessary conditions on $\mathcal{M}$ if it is to have a parallel decomposition.

Now, the injectivity of the state map forces the induced state partitions to be orthogonal. This is because each state in $\mathcal{M}$ corresponds to precisely one state in $\mathcal{M}_{\|}$ which in turn corresponds to precisely one state in each of the component machines.

So, for the ROBOT-TESTER-FACTORY example, both the state and clock maps are identity functions, and the induced state partitions are given by

$$\begin{aligned} \pi_R \;=\; & \{\, \{(1,A),(1,B),(1,C)\} \\ & \{(2,A),(2,B),(2,C)\} \\ & \{(3,A),(3,B),(3,C)\} \\ & \{(4,A),(4,B),(4,C)\} \,\} \end{aligned}$$

and

$$\begin{aligned} \pi_T \;=\; & \{\{(1,A),(2,A),(3,A),(4,A)\} \\ & \{(1,B),(2,B),(3,B),(4,B)\} \\ & \{(1,C),(2,C),(3,C),(4,C)\}\} \end{aligned}$$

Taking an arbitrary member from each of π_R and π_T we see that their intersection is the singleton corresponding precisely to the product of the states that gave rise to those members. Excepting the relabelling and structural inclusion this is representative.

Lemma 4.13

The induced state partitions, given in Definition 4.11, are orthogonal.

Proof:

Because ϕ is one-to-one its inverse is a partial function. Thus,

$$\begin{aligned}
\pi_1.\pi_2 &= \{U \cap V | U \in \pi_1, V \in \pi_2\} && \textit{Section } 2.1.1 \\
&= \{\phi^{-1}[(q_1, Q_2)] \cap \phi^{-1}[(Q_1, q_2)] | q_1 \in Q_1, q_2 \in Q_2\} && \textit{Definition } 4.11 \\
&= \{\phi^{-1}[\{(q_1, q_2)\}] | (q_1, q_2) \in Q_\parallel\} \\
&= \{\phi^{-1}[\{q_\parallel\}] | q_\parallel \in Q_\parallel\} && \textit{Definition } 4.2 \\
&= \bot_Q && \textit{because } \phi \textit{ is total}
\end{aligned}$$

□

The intuitions behind the above proof are as follows. A member of the first state partition consists of states in the original machine that have the same first component when relabelled by ϕ, and similarly for the second state partition. Because the relabelling was one-to-one, the pair of members—one from each induced state partition—defined by a given state in the original machine is unique, and each state is contained in both members, so that the partitions are orthogonal.

■□■

If either of the state partitions were trivial then one of them would have to be $\bot_Q$ contradicting the assumption that $|Q_1|, |Q_2| < |Q|$. Notice that neither π_R or π_T are trivial.

Lemma 4.14

The induced state partitions are non-trivial.

Proof:

Assume one of π_1 and π_2, WLOG π_1, is trivial. Either it is $\top_Q$ in which case $\pi_2 = \bot_Q$ (because $\pi_1.\pi_2 = \bot_Q$), or it is $\bot_Q$. In either case at least one of π_1 and π_2 is equal to $\bot_Q$. WLOG, assume

$\pi_1 = \perp_Q$. Now because Q is finite we have

$$\begin{aligned} |Q| &= |\perp_Q| && \textit{defn of } \perp \\ &= |\pi_1| && \textit{by assumption} \\ &= |\{\phi^{-1}[(q_1, Q_2)] | q_1 \in Q_1\}| && \textit{Definition } 4.11 \\ &< |\{(\phi \circ \phi^{-1})[(q_1, Q_2)] | q_1 \in Q_1\}| \\ &= |\{(q_1, Q_2) | q_1 \in Q_1\}| \\ &= |Q_1| \end{aligned}$$

contradicting the assumption that $|Q_1| < |Q|$.

□

The intuition behind the proof is as follows. The state size requirement for a parallel decomposition forces at least some states to have the same first (equivalently second) component when relabelled by ϕ. The definition of the induced state partitions, Definition 4.11, gives immediately that this is equivalent to the partitions being non-trivial.

■□■

Now, in a similar manner to before, there is a requirement that forces the appropriate partitions to induce a well-defined transition function. The crucial factor here is that in creating component machines one can join states together in an arbitrary manner but there is no guarantee that determinacy will follow.

For example, if $(q, \nu) \xrightarrow{a,t} (r, \xi)$ and $(q', \nu) \xrightarrow{a,t} (r', \xi')$ where q and q' are to be joined together whilst r and r' are not then the new machine is non-deterministic. More specifically, $(\{q, q'\}, \nu) \xrightarrow{a,t} (r, \xi)$ and $(\{q, q'\}, \nu) \xrightarrow{a,t} (r', \xi')$ where $\{q, q'\}$ is the new state obtained by combining q and q'.

As a more specific illustration, suppose we have a large timed machine, some of whose edges are shown in Figure 4.7.

If q_0 and q_1 are to be joined together, the resulting state will represent the fact that the machine is in one of states q_0 or q_1, but provides no information giving precisely which one. This sort of joining of states is all that we are allowing.

Consider the situation where the machine in question is in one of states q_0 or q_1, and the next input is an "a". The possible transitions are depicted in Figure 4.8.

If the value of x is 4 then the machine in question will be move to one of states q_4 or q_6, but without the knowledge of which of states q_0 or q_1 the machine is in, it is impossible to determine which of q_4 or q_6 the

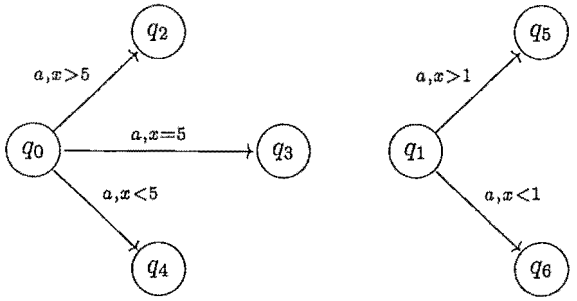

Figure 4.7: Part of a large machine.

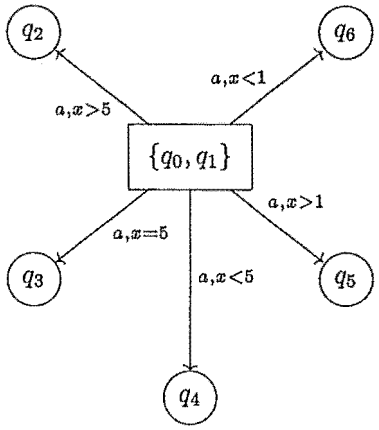

Figure 4.8: Potential transitions of large machine.

machine will be in. Thus states q_4 and q_6 must be joined together as well. This is depicted in Figure 4.9.

Suppose instead that x had value 5. Now the machine would move to states q_3 or q_5. Again, without the knowledge of which of states q_0 or q_1 the machine is actually in, it is impossible to determine which of states q_3 or q_5 the machine will be in. Thus they, states q_3 and q_5, must be joined together as well. Similarly, if x had value 0.5, then states q_4 and q_6 are indistinguishable, and so must be joined as well. Likewise, considering x having value 6 shows that states q_2 and q_5 must be joined as well.

Thus, all of q_2, q_3, q_4, q_5 and q_6 must be joined together. Therefore whatever the value of x, the machine will move from one of $\{q_0, q_1\}$ to one of $\{q_2, q_3, q_4, q_5, q_6\}$. The edge constraint will be $(x > 5) \vee (x = 5) \vee (x < 5) \vee (x > 1) \vee (x < 1)$ which is equivalent to T. The resulting forced joining is shown in Figure 4.10.

The reader may notice that we are ignoring some information by restricting ourselves to only this sort of joining, which carries forward no information as to the current value of clocks. This is indeed the case, and allowing clock information to be passed on results in a different decomposition result. A possible

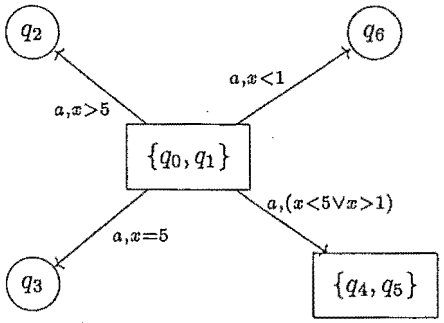

Figure 4.9: Potential transitions of large machine.

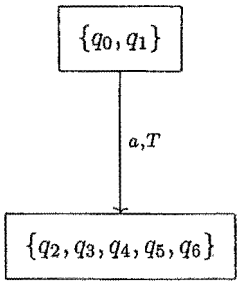

Figure 4.10: Potential transitions of large machine.

variation based on this is discussed briefly in the conclusion.

In the more general instance, if some extended states are to be joined, time determinacy requires for each symbol a and time step t, the extended states that could be reached by a transition on an a taking time t must also be joined. This motivates the following definition.

Definition 4.15

If we have a timed machine, $\mathcal{M}_*$, a partition $\pi \in \Pi(Q_*)$, and a set of stopwatches, $S \subseteq C_*$, then we say that (π, S) is an admissible pair *for $\mathcal{M}_*$ if the following condition is satisfied.*

For every pair of edges $e, e' \in E$ of the form $e = (q, r, a, X, \delta), e' = (q', r', a, X', \delta')$ if $\pi|_q = \pi|_{q'}$ and $(\delta \wedge \delta')|_S$ is satisfiable over $(\mathbb{R}^+)^S$ then $\pi|_r = \pi|_{r'}$ and $X \cap S = X' \cap S$.

Because we will want to talk about deterministic machines, and determinism is defined in terms of the transition relation, for most of the discussions we will use the following equivalent definition which rephrases admissibility in terms of the transition relation. For the example detailed in Figure 4.7 through Figure 4.10, we might have used the following argument for demonstrating that states $\{q_2, q_3, q_4, q_5, q_6\}$ must be joined if q_0 and q_1 are. Let time states be denoted by the value of x in that time state.

Because $(q_0,3) \xrightarrow{a,1} (q_4,4)$ and $(q_1,3) \xrightarrow{a,1} (q_5,4)$ we know that q_4 and q_5 must be joined. Similarly $(q_0,3) \xrightarrow{a,2} (q_3,5)$ and $(q_1,3) \xrightarrow{a,2} (q_4,5)$ imply that q_3 and q_4 must be joined. Appropriate additional transitions show the remaining forced joinings as before.

Lemma 4.16

If we have a timed machine, $\mathcal{M}_*$, a partition $\pi \in \Pi(Q_*)$, and a set of stopwatches, $S \subseteq C_*$, then (π, S) is an admissible pair *for $\mathcal{M}_*$ iff the following condition is satisfied.*

For every $q, q' \in Q_*$ and $\nu, \nu' \in (\mathbb{R}^+)^{C_*}$, if

$$(q,\nu) \xrightarrow{a,t} (r,\xi) \text{ and } (q',\nu') \xrightarrow{a,t} (r',\xi')$$

with $\pi|_q = \pi|_{q'}$ and $\nu|_S = \nu'|_S$ then $\pi|_r = \pi|_{r'}$ and $\xi|_S = \xi'|_S$.

Proof:

We first demonstrate that admissibility implies the condition in the hypothesis, before demonstrating the converse. If (π, S) is an admissible pair for $\mathcal{M}_*$ then for every $q, q' \in Q_*$ and $\nu, \nu' \in (\mathbb{R}^+)^{C_*}$ if

$$(q,\nu) \xrightarrow{a,t} (r,\xi) \text{ and } (q',\nu') \xrightarrow{a,t} (r',\xi')$$

with $\pi|_q = \pi|_{q'}$ and $\nu|_S = \nu'|_S$ then by the definition of transition (Definition 2.32), there must be edges of the form $e = (q, r, a, X, \delta)$ and $e' = (q', r', a, X', \delta')$ in E so that $\nu + t \models \delta$, $\nu' + t \models \delta'$ with $\xi = [X \to 0](\nu + t)$ and $\xi' = [X' \to 0](\nu' + t)$. Now because $\nu \models \delta$, from Proposition 2.4 it follows that $\nu|_S \models \delta|_S$. Similarly $\nu'|_S \models \delta'|_S$, so that $\delta|_S \wedge \delta'|_S$ is satisfiable, and hence $(\delta \wedge \delta')|_S$ is satisfiable over $(\mathbb{R}^+)^S$. Admissibility now requires that $\pi|_r = \pi|_{r'}$ and $X \cap S = X' \cap S$. Because $X \cap S = X' \cap S$ and $\nu|_S = \nu'|_S$ we know that

$$\begin{aligned}
\xi|_S &= ([X \to 0](\nu + t))|_S \\
&= [X \cap S \to 0](\nu + t)|_S \\
&= [X \cap S \to 0](\nu|_S + t) \\
&= [X' \cap S \to 0](\nu'|_S + t) \\
&= [X' \cap S' \to 0](\nu' + t)|_S \\
&= ([X' \to 0](\nu' + t))|_S \\
&= \xi'|_S
\end{aligned}$$

which together with $\pi|_r = \pi|_{r'}$ demonstrates the equivalence in one direction.

For the converse, assume the condition stated in the lemma holds. Take an arbitrary pair of edges $e, e' \in E$ of the form $e = (q, r, a, X, \delta), e' = (q', r', a, X', \delta')$ where $\pi|_q = \pi|_{q'}$ and $(\delta \wedge \delta')|_S$ is satisfiable.

Because $(\delta \wedge \delta')|_S$ is satisfiable over $(\mathbb{R}^+)^S$, there is some $\nu_S \in (\mathbb{R}^+)^S$ so that $\nu_S \models (\delta \wedge \delta')|_S = \delta|_S \wedge \delta'|_S$. This means that $\nu_S \models \delta|_S$ and $\nu_S \models \delta'|_S$.

Now, take an arbitrary $t < t_\nu = \min_{x \in S}\{\nu_S(x)\}$, and define $\nu \in (\mathbb{R}^+)^C$ by $\nu|_S = \nu_S$ and $\nu|_{C-S} = t_\nu$, the constant function on $C - S$. From Proposition 2.4 this means $\nu \models \delta$, and $\nu \models \delta'$. By the definition of a transition, because $(\nu - t) + t \models \delta$ (t being less than t_ν guarantees that $\nu - t \in (\mathbb{R}^+)^C$) we have that $(q, \nu - t) \xrightarrow{a,t} (r, \xi)$ where $\xi = [X \to 0]((\nu - t) + t)$. Similarly $(q', \nu - t) \xrightarrow{a,t} (r', \xi')$ where $\xi' = [X' \to 0]((\nu - t) + t)$. Now because $\pi|_q = \pi|_{q'}$ and the two time states are equal the assumption that the condition stated in the lemma holds then guarantees that $\pi|_r = \pi|_{r'}$ and $\xi|_S = \xi'|_S$. Now

$$\begin{aligned} \xi|_S &= ([X \to 0]((\nu - t) + t))|_S \\ &= [X \cap S \to 0]\nu|_S \\ &= [X \cap S \to 0]\nu_S \end{aligned}$$

and similarly $\xi'|_S = [X' \cap S \to 0]\nu_S$. Because $\nu_S \in (\mathbb{R}^+)^S$ this means that $X \cap S = X' \cap S$ which together with $\pi_r = \pi|_{r'}$ completes the proof.

□

Another way of viewing this description of admissible pairs is as follows. Given a member of the partition, U, and an (action,time state) pair, (a, t), any two transitions from a member of U on a and t must both go into the same member of π if the time states have the same values for members of S. This means we can find admissible partitions as a recursive fixed point because the system is finite. Indeed $(\bot_Q, C)$ and $(\top_Q, S)$ for any $S \subseteq C$, are admissible pairs for any machine $\mathcal{M}$.

■□■

Note that both (π_R, C_R) and (π_T, C_T) are admissible pairs of $\mathcal{M}_F$, and that they generate the transition structures in Figure 4.11 and Figure 4.12. Notice that the transition structures induced in this manner are identical to those of the respective component machines and, importantly, they are deterministic.

The reason we define admissible pairs is that in order to generate the component machines from $\mathcal{M}$ we will need to group some states of $\mathcal{M}$ in precisely this sort of a consistent manner, as the following result shows.

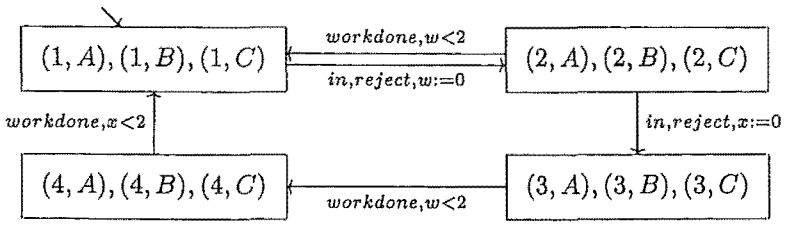

Figure 4.11: Partition of FACTORY induced by ROBOT

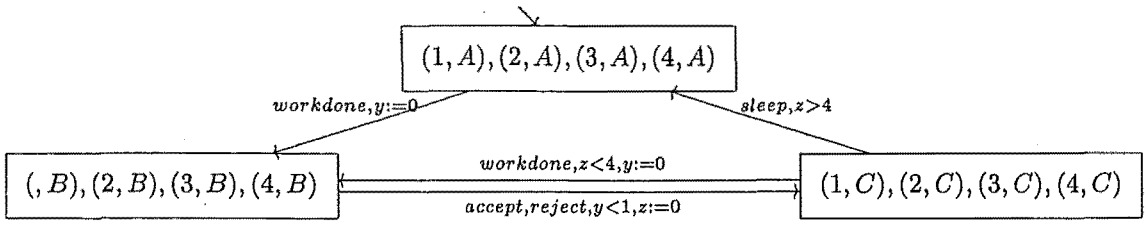

Figure 4.12: Partition of FACTORY induced by TESTER

Lemma 4.17

If $\mathcal{M}$ has a parallel decomposition with induced state partitions π_1 and π_2 and induced clock partition (S_1, S_2) then (π_1, S_1') and (π_2, S_2') are admissible where the subscripts refer to the component which generated the associated partition on $\mathcal{M}$.

Proof:

Let the decomposition be realised by $(\phi, \eta) : \mathcal{M} \trianglelefteq \mathcal{M}_1 || \mathcal{M}_2$ where for each of $i = 1, 2$ we have that π_i and C_i are induced by $\mathcal{M}_i$. Consider the case for (π_1, S_1). Now, for each $a \in \Sigma$ and $t \in \mathbb{R}^+$, take $q, q' \in U \in \pi_1$ and $\nu, \nu' \in (\mathbb{R}^+)^{C_*}$ where $\nu|_{S_1} = \nu'|_{S_1}$.

If $(q, \nu) \xrightarrow{a,t} (r, \xi)$ and $(q', \nu') \xrightarrow{a,t} (r', \xi')$ then because (ϕ, η) is a homomorphism, we know that

$$(\phi(q), \nu \circ \eta^{-1}) \xrightarrow{a,t} (\phi(r), \xi \circ \eta^{-1})$$

in $\mathcal{M}_1 || \mathcal{M}_2$. If we let $\phi(q) = (q_1, q_2)$ and $\phi(r) = (r_1, r_2)$ then

$$((q_1, q_2), (\nu \circ \eta^{-1})|_{C_1} \uplus (\nu \circ \eta^{-1})|_{C_2}) \xrightarrow{a,t} ((r_1, r_2), (\xi \circ \eta^{-1})|_{C_1} \uplus (\xi \circ \eta^{-1})|_{C_2})$$

in $\mathcal{M}_1 || \mathcal{M}_2$ so that, from Lemma 4.4

$$(q_1, (\nu \circ \eta^{-1})|_{C_1}) \xrightarrow{a,t} (r_1, (\xi \circ \eta^{-1})|_{C_1})$$

in $\mathcal{M}_1$. Similarly we find that

$$(q_1', (\nu' \circ \eta^{-1})|_{C_1}) \xrightarrow{a,t} (r_1', (\xi' \circ \eta^{-1})|_{C_1})$$

in $\mathcal{M}_1$ where $\phi(q') = (q_1', q_2')$ and $\phi(r') = (r_1', r_2')$. But because $U \in \pi$, U is of the form $\phi^{-1}[(q_1^*, Q_2)]$ for some $q_1^* \in Q_1$ so that $q_1 = q_1^* = q_1'$. Also we have that

$$\begin{aligned}(\nu \circ \eta^{-1})|_{C_1} &= \nu|_{S_1} \circ \eta^{-1} \\ &= \nu|_{S_2} \circ \eta^{-1} \\ &= (\nu \circ \eta^{-1})|_{C_2}\end{aligned}$$

by hypothesis that $\nu|_{S_1} = \nu'|_{S_1}$ and because $S_1 = \eta^{-1}[C_1]$ by definition of induced clock partition.

Thus $(q_1, (\nu \circ \eta^{-1})|_{C_1})$ and $(q_1', (\nu' \circ \eta^{-1})|_{C_1})$ are identical. Determinism of $\mathcal{M}_1$ now gives that $(r_1, (\xi \circ \eta^{-1})|_{C_1})$ and $(r_1', (\xi' \circ \eta^{-1})|_{C_1})$ must be identical also.

This gives $r_1 = r_1'$ so that $r, r' \in \phi^{-1}[(r_1, Q_2)] \in \pi_1$, that is $\pi_1|_r = \pi_1|_{r'}$. Also $(\xi \circ \eta^{-1})|_{C_1} = (\xi' \circ \eta^{-1})|_{C_1}$ so that $\xi|_{\eta^{-1}[C_1]} = \xi'|_{\eta^{-1}[C_1]}$. S_1 is by definition $\eta^{-1}[C_1]$, so that $\xi|_{S_1} = \xi'|_{S_1}$

Thus (π_1, S_1) is admissible by Lemma 4.16. The argument for (π_2, S_2) is similar.

□

Informally, given the decomposition, with monomorphism (ϕ, η), we take an arbitrary pair of transitions T, T' of $\mathcal{M}$ whose initial time states are identical over S_1 and whose initial states come from the same member of π_1, that is the first component of their images under ϕ are identical.

The definition of homomorphism gives corresponding transitions in $\mathcal{M}_1 \| \mathcal{M}_2$, with the images of the states under ϕ and the corresponding time states (Figure 4.5). Lemma 4.4 now relates the transitions in each of $\mathcal{M}_1$ and $\mathcal{M}_2$. We consider those on $\mathcal{M}_1$ only.

The initial requirements on T and T' ensure that the initial extended states of the two (one for each of T and T') corresponding transitions on $\mathcal{M}_1$ are identical. Time determinacy of $\mathcal{M}_1$ now forces the final extended states to also be identical. Thus the final states of T and T' are in the same member of π_1 and the final time states are identical over $S_1 = \eta^{-1}[C_1]$. An analogous argument for the second component gives the result.

■□■

Admissible pairs are used in the same way as SP-partitions were in Chapter 3. We define quotient machines in a similar manner. The constraint definition is motivated by the earlier discussion from Figure 4.7 to Figure 4.10, and Definition 4.15.

Definition 4.18

If we have a timed machine $\mathcal{M}$ and π is a partition of Q, with $S \subseteq C$, then we define the quotient machine *of $\mathcal{M}$ with (π, S) to be the machine $\mathcal{M}_\pi$ where :*

1. $Q_\pi = \pi$,
2. $\Sigma_\pi = \Sigma$,
3. $s_\pi = \pi|_s$,
4. $C_\pi = S$,
5. *E_π is defined so that $e_\pi \in E_\pi$ iff $e_\pi = (\pi|_q, \pi|_r, a, X \cap S, \delta_\pi)$ for some $e = (q, r, a, X, \delta) \in E$, where δ_π is the minimal fixed point of the recursion*

$$\delta_\pi = \delta|_S \vee \left(\bigvee \{ \delta'|_S | (q', r', a, X', \delta') \in E, \pi|_{q'} = \pi|_q \text{ and } \delta'|_S \wedge \delta_\pi \text{ is satisfiable } \} \right)$$

   The starting point for the recursion is $\delta_\pi = F$. Minimal here is with the respect to the partial order generated by implication, $\Rightarrow$.

6. $\mathcal{F}_\pi = \pi|_{\mathcal{F}}$.

The machine $\mathcal{M}_\pi$ is denoted by $M/(\pi, S)$

Proposition 4.19

The starting point, $\delta_\pi = F$ for the quotient edge recursion generates the unique mimimal fixed point.

Proof:

The existence of a fixed point is guaranteed because there are only finitely many edges in $\mathcal{M}$, and once an edge is included it remains.

For the minimality let the fixed point generated by iterating the recursion be δ_*. Take an arbitrary function, f, modelling δ_*, then because f cannot model F, it must model δ'_S for some edge e', by the definition of $\wedge$. Because of the recursion, there must be a sequence of edges from e to e', such that the constraints of any two consecutive edges are mutually satisfiable when restricted to S (that is, $(\delta_i \wedge \delta_{i+1})|_S$ is satisfiable for each index i). Because of this each of the edges in the sequence must be present in any expression fulfilling the recursion, so that f will model any expressing satisfying the recursion. Thus, δ_* is minimal.

■□■

Proposition 4.20

If e is an edge in the machine $\mathcal{M}$ and e_π is the edge in $\mathcal{M}/(\pi, S)$ obtained from e, then $\delta|_S \Rightarrow \delta_\pi$.

Proof:

Immediate fron the definition of disjunction, $\wedge$.

■□■

For the ROBOT-TESTER-FACTORY example, the quotient machines generated by the two admissible pairs are the machines whose states and transition structures are described in Figure 4.11 and Figure 4.12.

The process of grouping together states and ignoring clocks, then considering the induced edge structure led to the concept of admissibility. Determinacy in the edge structure thus generated requires admissibility. This inducing of edge structures is encapsulated by the forming of quotient machines, and so deterministic quotient machines will require admissable pairs. What is not as transparent is the fact that the argument used to show the deterministic quotient machines require admissible pairs is reversible, so that admissible pairs always generate deterministic quotient machines.

Lemma 4.21

$\mathcal{M}/(\pi, S)$ is deterministic iff (π, S) is admissible.

Proof:

Consider two arbitrary edges of $\mathcal{M}_\pi = \mathcal{M}/(\pi, S)$, $e_\pi = (q_\pi, r_\pi, a, X_\pi, \delta_\pi)$ and $e'_\pi = (q'_\pi, r'_\pi, a, X'_\pi, \delta'_\pi)$, with the same action and $q_\pi = q'_\pi$.

Determinism of $\mathcal{M}_\pi$ requires that if there is some time state, ν_π, so that $\nu_\pi \models \delta_\pi$ and $\nu_\pi \models \delta'_\pi$ then $e_\pi = e'_\pi$ because both edges can be traversed from (q_π, ν_π). This means that if $\delta_\pi \wedge \delta'_\pi$ is satisfiable then $e_\pi = e'_\pi$.

Now, e_π and e'_π, by virtue of their being edges of a quotient machine, are of the form $e_\pi = (\pi|_q, \pi|_r, a, X \cap S, \delta|_\pi)$ and $e'_\pi = (\pi|_{q'}, \pi|_{r'}, a, X' \cap S, \delta|'_\pi)$ for some $(q, r, a, X, \delta), (q', r', a, X', \delta') \in E$

Note that $\pi|_q = q_\pi = q'_\pi = \pi|_{q'}$ by the choice of edges above and $(\delta \wedge \delta')|_S = \delta|_S \wedge \delta'|_S \Longrightarrow \delta_\pi \wedge \delta'_\pi$ by Definition ?? . Determinism thus requires that if $(\delta \wedge \delta')|_S$ is satisfiable then $e_\pi = e'_\pi$ which means that both $r_\pi = r'_\pi$ and $X_\pi = X'_\pi$, or alternatively both $\pi|_r = \pi|_{r'}$ and $X \cap S = X' \cap S$ which is the requirement for (π, S) to be admissible.

For the converse, if $\mathcal{M}_\pi$ is not deterministic then there is some extended state of $\mathcal{M}_\pi$ which can make two distinct transitions, that is satisfy the requirements of constraints from two distinct edges. Writing

everything in terms of the original machine as above, this means it can't be the case that both $\pi|_r = \pi|_{r'}$ and $X \cap S = X' \cap S$. This means that (π, S) is not admissible as required.

■□■

We are now coming to the most important part of this chapter. Admissibility was arrived at as a necessary requirement for deterministic quotient machines, where our concept of quotient machines arises by grouping states together and ignoring clocks. With relabellings, this is certainly true of components of parallel machines. Each state of a component of a parallel machine corresponds to many states in the parallel machine, and the clock set of a component machine is contained in that of the parallel machine it is a component of. This means that to go from the parallel machine to a component of it requires grouping together states, and ignoring clocks and this is exactly what quotient machines do.

Once the quotient machine has been generated there is a natural correspondence between the original machine and its quotient. That is, map each state of the original machine to the state of the quotient machine which contains it. This is a well defined map since the states of the quotient machine are disjoint groups of states from the original machine. Similarly, since the clock set of the quotient machine is contained in the clock set of the original machine, there is a natural partial map between the two, that is, the identity restricted to the clocks of the quotient machine. Because the quotient machine preserves the edges of the original machine, these natural maps are in fact epimorphisms. This result, when combined with Lemma 4.9 provides sufficient conditions to match the necessary ones on the induced state and clock partitions, Definition 4.11 and Definition 4.12, which were demonstrated earlier in the section.

Lemma 4.22

For an arbitrary quotient machine, $\mathcal{M}/(\pi, S)$, $\mathcal{M}$ is epimorphic to $\mathcal{M}/(\pi, S)$.

Proof:

Let $\mathcal{M}_\pi = \mathcal{M}/(\pi, S)$.

Define $\phi \in Q_\pi^Q$ by $\phi(q) = \pi|_q$, and $\eta : C \to C_\pi$ by $\nu = \{(c, c) | c \in C_\pi\}$. Then,

1. $\phi(s) = \pi|_s = s_\pi$, from Definition 4.18.

2. If $(q, \nu) \xrightarrow{a,t} (r, \xi)$ in $\mathcal{M}$ then we know $(q, r, a, X, \delta) \in E$ for some $(X, \delta) \in (2^C, \Phi(C))$ where $\nu + t \models \delta$ and $\xi = [X \to 0](\nu + t)$, from Definition 2.32. Thus, $(\pi|_q, \pi|_r, a, X \cap X, \delta_\pi) \in E_\pi$ where $\delta|_S \Rightarrow \delta_\pi$

Now $\nu \circ \eta^{-1} + t = \nu|_S + t = (\nu + t)|_S \models \delta|_S$, so that $(\pi_q, \nu|_S) \xrightarrow{a,t} (\pi_r, \xi')$ in $\mathcal{M}_\pi$ where

$$\begin{aligned} \xi' &= [X \cap S \to 0](\nu|_S + t) \\ &= [X \cap S \to 0]((\nu + t)|_S) \\ &= ([X \to 0](\nu + t))|_S \\ &= \xi|_S \end{aligned}$$

Thus, from Definition 4.18 $(\phi(q), \nu \circ \eta^{-1}) \xrightarrow{a,t} (\phi(r), \xi \circ \eta^{-1})$ in $\mathcal{M}_\pi$.

3. $\phi[\mathcal{F}] = \pi|_{\mathcal{F}} = \mathcal{F}_\pi$, from Definition 4.18.

Thus (ϕ, η) is a homomorphism.

Now, because no member of π is empty, ϕ is onto. Similarly, since $S \subset X$, η is onto, and thus (ϕ, η) is a monomorphism as required.

□

We define the maps ϕ and η, so that ϕ maps each state of $\mathcal{M}$ to the state of $\mathcal{M}/(\pi, S) = \mathcal{M}_\pi$ that it is a member of $(Q_\pi = \pi)$, and η is the identity map with its domain restricted to the clocks of $\mathcal{M}_\pi$. Thus, the clock map is in general only a partial map whilst the state map is total.

Conditions 1 and 3 of (ϕ, η) being a homomorphism are satisfied immediately via the definition of a quotient machine.

For the second condition take an arbitrary transition in $\mathcal{M}$, from extended state (q, ν) to (r, ξ). This corresponds to traversing an edge $e \in E$ whose constraint is satisfied at some intermediate time step. By the definition of a quotient machine there is an edge e_π corresponding to e. The extended state of $\mathcal{M}_\pi$ corresponding to (q, ν) is $(\phi(q), \nu \circ \eta^{-1})$ and satisfies the criteria for traversing e_π.

The resulting extended state is expressible in terms of the initial extended state $(\phi(q), \nu \circ \eta^{-1})$ and the components of e_π via the definition of a transition. The form of e_π immediately guarantees that this resulting state corresponds to r. That the time states correspond is verified by checking each clock in C_π.

Thus the second and final condition for (ϕ, η) being a homomorphism is fulfilled. That both ϕ and η are onto is easily checked from their definitions, so that (ϕ, η) is the required monomorphism.

■□■

This is enough to give us the desired theorem.

Theorem 4.23

A timed machine $\mathcal{M}$ has a parallel decomposition iff it has two orthogonal non-trivial state partitions, π_1 and π_2, and there is a clock partition, $\{C_1, C_2\}$, so that both (π_1, C_1) and (π_2, C_2) are admissible.

Proof:

If $\mathcal{M}$ has a parallel decomposition, then let π_1 and π_2 be the induced state partitions defined in Definition 4.11, and let C_1 and C_2 be defined so that $\{C_1, C_2\}$ is the induced state partition defined in Definition 4.12.

Then, by Lemma 4.13, π_1 and π_2 are orthogonal. Also, by Lemma 4.14 they are non-trivial, and by Lemma 4.17 both (π_1, C_1) and (π_2, C_2) are admissible. Thus, constructively, $\mathcal{M}$ has two orthogonal non-trivial state partitions, π_1 and π_2, and there is a clock partition, $\{C_1, C_2\}$, so that both (π_1, C_1) and (π_2, C_2) are admissible.

For the converse, suppose that $\mathcal{M}$ has two orthogonal non-trivial state partitions, π_1 and π_2, and there is a clock partition, $\{C_1, C_2\}$, so that both (π_1, C_1) and (π_2, C_2) are admissible. Let $\mathcal{M}_1 = \mathcal{M}/(\pi_1, C_1)$ and $\mathcal{M}_2 = \mathcal{M}/(\pi_2, C_2)$. Now, from Lemma 4.21, both $\mathcal{M}_1$ and $\mathcal{M}_2$ are deterministic. Also, from Lemma 4.22, $\mathcal{M}$ is epimorphic to both $\mathcal{M}_1$ and $\mathcal{M}_2$. In particular, it is homomorphic to the two quotient machines, which from Lemma 4.9 means that it is homomorphic to their parallel composition. The orthogonality of π_1 and π_2 ensures that the state map is injective, so that $\mathcal{M}$ is monomorphic to the parallel composition of $\mathcal{M}_1$ and $\mathcal{M}_2$. The non-triviality of π_1 means that $|Q_1| = |\pi_1| < |Q|$, and similarly $|Q_2| < |Q|$ so that $\mathcal{M}$, by Definition 4.10, has a parallel decomposition.

■□■

In a similar manner to Chapter 3, it is possible to extend the definition of parallel composition to many components, and likewise the definition of a parallel decomposition. In a manner entirely similar to that used in Chapter 3 the many component case reduces to the two component one using a lattice of admissible pairs. The $\wedge$ of two admissible pairs involves taking the product of the state partitions, and the disjoint union of the two clock sets. Thus Definition 4.10 can be strengthened to many components, although we do not go through the details here.

4.3.1 Examples

As an illustration of the above result, we will go through two complete examples which will demonstrate the application of the above results, and also the case when the structural inclusion map is not the identity (or indeed an isomorphism).

Consider the picture in Figure 4.13. It describes the operation of a buffer of size two. The buffer can be reset after processing one or two inputs, but the buffer can only perform one push (or pop) per unit of time. This is a modification of the example from [Zie87] used in Chapter 3.

The actions intuitive meanings are as follows; "a" represents a push into the buffer, "b" represents a pop from the buffer, and "c" represents a clearing of the buffer. The buffer must be cleared at least once every two inputs processed, and successive "a"'s (or "b"'s) must be at least one time unit apart. Again, all cycles accepted. States 4 and 7 represent underflow and overflow states respectively.

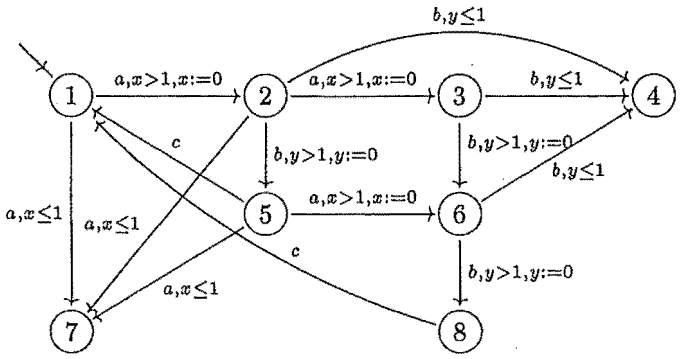

Figure 4.13: A "big" timed machine - BUFF2

In order to find appropriate admissible pairs it is going to be necessary to partition the state set twice (once for each admissible pair) and the clock set once (into two sets, one for each admissible pair). Looking at just the clock set, we note that one of the two admissible pairs will not contain the clock y. Now because there is a transition on a b out of state 2 in to each of states 4 and 5, and each of the constraints restricted to $C - \{y\}$ is T, then states 4 and 5 must be grouped together. This is shown in Figure 4.14.

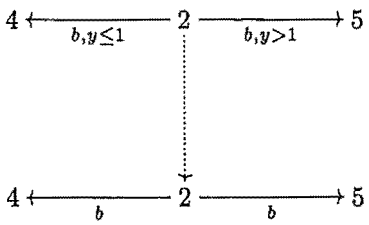

Figure 4.14: Trying to find an admissible pair for BUFF2

In a similar manner states 4 and 6 must be grouped together because of the b transitions out of state 3. Similarly states 4 and 8 must be grouped together because of the b transitions out of state 6. Thus states 4, 5, 6 and 8 must all be grouped together. This generates the picture in Figure 4.15.

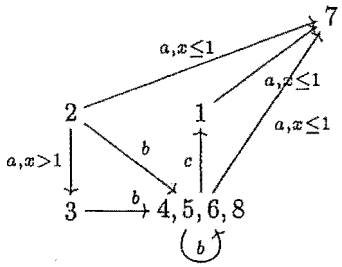

Figure 4.15: Trying to find an admissible pair for BUFF2

In Figure 4.15 we observe that the grouping above, along with $\{x\}$ forms an admissible pair, and the reasoning used to generate this ensures that there is no smaller partition (using the partial ordering defined in Definition 2.5) that when coupled with the set $\{x\}$ generates an admissible pair. Denote this partition π_1.

In addition, one of the two admissible pairs will not contain the clock x. This would mean that states 2 and 7 must be grouped together because the constraints on the a transitions out of state 1 both become T when restricted to $C - \{x\}$. In fact restricting every constraint to be over $C - \{x\}$ gives the picture in Figure 4.16.

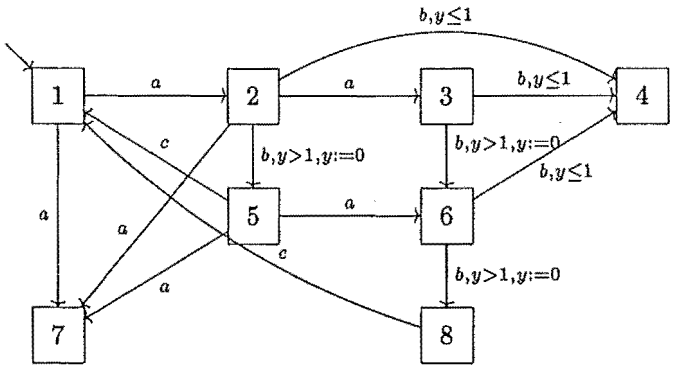

Figure 4.16: BUFF2 with constraints restricted to $C - \{x\}$

This shows that states 2 and 7 must be grouped. Similarly, states 3 and 7 must be grouped because of the a transitions out of state 2. The a transitions out of state 5 force states 6 and 7 to be grouped as well, giving the picture in Figure 4.17 where β stands for $b, y > 1, y := 0$. The three β transitions out of the group $2, 3, 6, 7$ arise because of the transitions between states 2 and 5; 3 and 6; and 6 and 8 from BUFF2.

The three β transitions require that states 5 and 8 must also be grouped with 2,3,6, and 7, giving the picture in Figure 4.18. with $q_0 = \{1\}$, $q_1 = \{2, 3, 4, 5, 7, 8\}$ and $q_2 = \{4\}$.

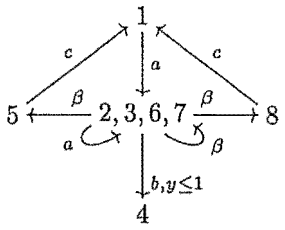

Figure 4.17: Trying to find an admissible pair for BUFF2

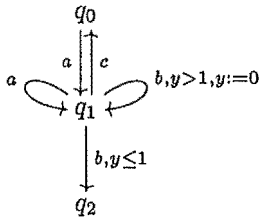

Figure 4.18: Along with $\{y\}$ forms an admissible pair for BUFF2

The groupings shown in Figure 4.18, along with $\{y\}$ form an admissible pair for BUFF2, and similarly with π_1 above, this is the smallest partition that when coupled with $\{y\}$ generates an admissible pair. Denote this partition π_2.

Because π_1 and π_2 are not orthogonal—their product still groups 5, 6, and 8—we know there can be no pair of orthogonal admissible pairs for BUFF2 where the two clock partitions are $\{x\}$ and $\{y\}$. Thus, if there is any pair of partitions to satisfy the hypothesis of Theorem 4.23, then the clock sets must be C and $\emptyset$.

The minimal partition, π_3, which when combined with $\emptyset$ forms an admissible pair, must be at least as big as $\pi_1 \vee \pi_2$ because anything grouped by either π_1, which ignored y, or π_2, which ignored x, must also be grouped by π_3. $\pi_1 \vee \pi_2$ groups every state except state 1, and together with $\emptyset$ forms an admissible partition, as shown in Figure 4.19 where $q_0 = \{1\}$ and $q_1 = \{2, 3, 4, 5, 6, 7, 8\}$.

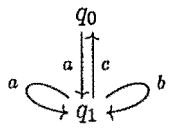

Figure 4.19: Along with $\emptyset$ forms an admissible pair for BUFF2

The other partition, π_4, will have the clock set $C = \{x, y\}$, but in order to satisfy the requirements of Theorem 4.23, it must be both non-trivial and orthogonal to π_3. This means that it must group some

states, and cannot group any of 2,3,4,5,6,7 or 8 together. Thus π_4 will be of the form $\top_{\{1,q\}} \wedge \bot_Q$ for some $q \in Q = \{1, 2, ..., 8\}$. Any $q \neq 2$ will generate an admissible pair. Figure 4.20 depicts the situation when $q = 7$.

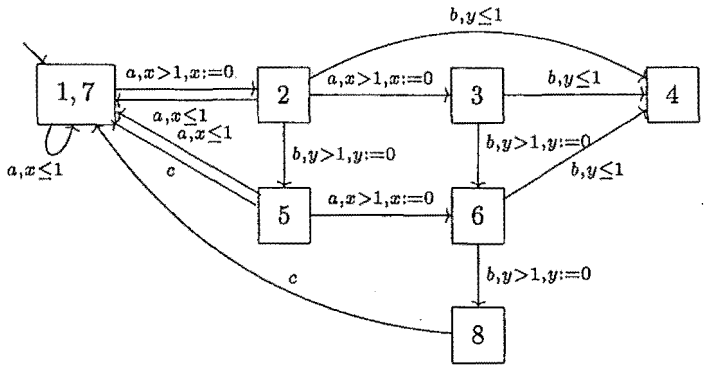

Figure 4.20: A partition orthogonal to the one in Figure 4.19

Thus BUFF2 does have a parallel decomposition—BUFF2 is contained in the parallel composition of BUFF2/$(\pi_3, \emptyset)$ and BUFF2/$(\pi_4, \{x, y\})$. This illustrates some of the limits of the result in Theorem 4.23. Finding the appropriate partitions may be very hard, and the decomposition requirement that the state sets of the components are smaller in cardinality than those of the "large" machine is not strict enough to truly accomplish our goal. BUFF2 had 8 states, and its two components have 2 and 7 states, so the total size of storing the two components is thus greater than that of BUFF2, even ignoring their transition structures. To get the components as small as possible requires finding the largest possible partitions fulfilling the requirements of Theorem 4.23. The example of BUFF2 as it stands does not enable us to see this properly, so we modify it slightly.

This concludes our discussion of this first example. Consider now another buffer of size 2, with the same states as BUFF2, but now each item cannot be "pop"ped within 1 time unit of it being "push"ed onto the buffer. Denote this machine REBUFF2, and it is depicted in Figure 4.21.

In a similar manner to that used for BUFF2, we see that any partition which will be combined with a clock set not containing w must group states 4, 5, and 6 together in order to form an admissible pair. Likewise, to form an admissible pair, any partition which will be combined with a clock set not containing x must group states 7 and 8 together. These groupings are orthogonal, and thus from Theorem 4.23 there is a parallel decomposition of REBUFF2. These two groupings are shown in Figure 4.23 and Figure 4.22.

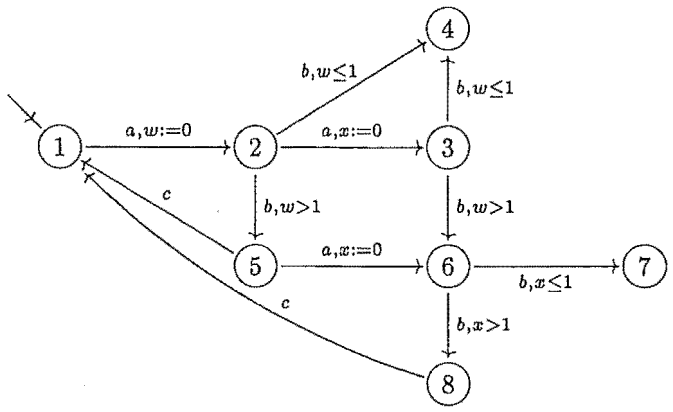

Figure 4.21: Another "big" timed machine - REBUFF2

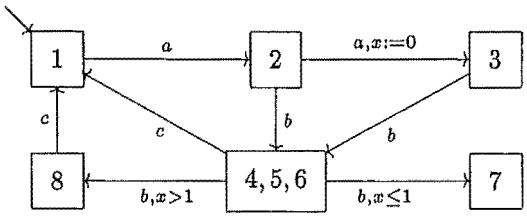

Figure 4.22: A partition that forms an admissible pair with $\{x\}$.

They are not, however, the largest orthogonal partitions that combine with $\{x\}$ and $\{w\}$ to form admissible pairs of REBUFF2. If we let π_5 and π_6 be defined as

$$\{\{1,8\},\{2,3,7\},\{4,5,6\}\}$$

and

$$\{\{1,3,5\},\{2,4\},\{6,7,8\}\}$$

respectively, then π_5 and π_6 are orthogonal, and also $(\pi_5,\{x\})$ and $(\pi_6,\{w\})$ are admissible pairs. This means that REBUFF2 will have a parallel decomposition, and in particular it will be monomorphic to (structurally contained in) the parallel composition of REBUFF2/$(\pi_5,\{x\})$ and REBUFF2/$(\pi_6,\{w\})$. These two components are shown in Figure 4.24. To find π_5 and π_6 we start with π_3 and π_4, and group together states one at a time maintaining admissibility. The choice of which states to group makes a difference to the size of the final components but this method will guarantee finding maximal (in the sense that there is no larger) admissible partitions. Notice that the two components both have size 3 whilst REBUFF2 has 8 states, so treating REBUFF2 as a parallel composition means that less states are required for storage, which is one of the original goals.

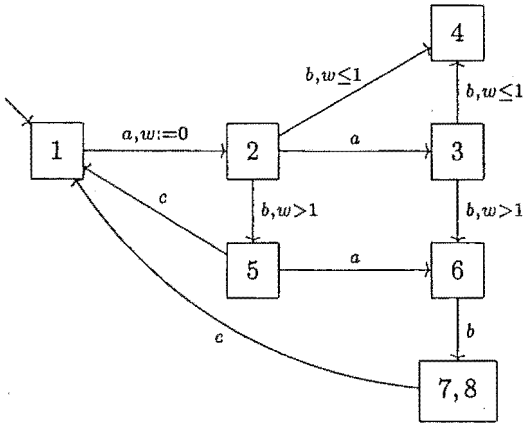

Figure 4.23: Partition that forms an admissible pair with $\{w\}$.

There is very little state redundancy in the parallel composition of the two components. The parallel machine has $3 \times 3 = 9$ states, where REBUFF2 has 8 states. In fact, the above example has the minimal state redundancy. Only the state $(\{1,8\},\{2,4\})$ does not correspond to a state in REBUFF2. There is, however, a lot of redundancy in the transitions. The parallel machine has a transition on a c from state $(\{1,8\},\{1,3,5\})$ to state $(\{1,8\},\{1,3,5\})$. State $(\{1,8\},\{1,3,5\})$ corresponds to state 1 in REBUFF2, and there is not a transition from state 1 to state 1 on a c in REBUFF2.

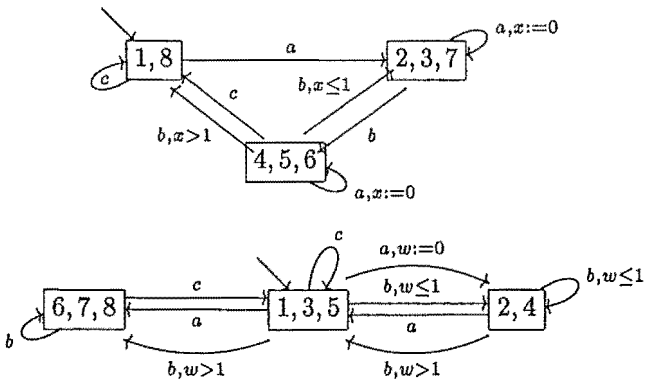

Figure 4.24: Two "small" components of REBUFF2. The first has only clock x and the second only clock w

This example provides both an illustration of how parallel decomposition can be used to decrease the state space, and shows how the components size can be at best the square root of the larger machine. It also illustrates, albeit informally, how one can go about finding admissible partitions, and some of the difficulties present in doing so. These difficulties become exacerbated when attempting to find minimal decompositions. Most importantly, however, these partitions give a feel for concepts such as grouping initial states, and then recursively seeing which states must be grouped together in order to satisfy the

definition of admissibility. The orthogonality requirement of Theorem 4.23 allows you to reject certain partitions during the process of working them out, and the clock partitions present natural starting points. This will hopefully help give some further insights as to both the motivations for the results and their application.

4.3.2 Summary

The work of this chapter extends the decomposition theory of Chapter 3 to timed machines. Although the results can be generalised, the requirements on state partitions are stricter, and thus there are fewer parallel decompositions for timed machines than for untimed machines. We first defined the concepts of parallel composition and structural inclusion, based on intuitive properties these "should" have. Some language theoretic consequences of these definitions are included to verify that the concepts as defined do indeed satisfy some of the properties desired. This provides sufficient foundation to pose formally the problem of when it is possible to break up a machine into components. A variety of necessary conditions follow, in the same vein as Chapter 3, showing that if a decomposition exists then so must partitions of the states and clocks with particular properties.

A notion of quotient machine is then introduced, and then a result showing that determinacy of a quotient requires that the state partitions and clock restrictions used to generate the quotient machine have many of the same properties as those arriving from parallel decompositions. The main result of the chapter is a naturally induced map from a machine to a quotient machine, which in turn shows that the necessary conditions provided earlier are in fact sufficient as well. Thus the problem of finding decompositions can be regarded as a problem of finding suitable state and clock partitions.

The chapter closed with two similar worked examples to demonstrate how the concepts of suitable partitions can be used to find decompositions of timed machines, and also illustrates some heuristics for finding these.

Chapter 5

Conclusion

5.1 Summary

Throughout the course of this work we have been reviewing, commenting on, and developing ways to treat a large abstract machine as a group of cooperating smaller, and hence more convenient to deal with, machines. The first task in our developing a theory of decomposition was to formalise a coherent concept of composition. In this, our motivation was that the composition would be of the lock-step parallel variety, meaning that each machine processes independently, but at the same rate. Whilst not being general enough to describe the typical notion of parallel composition whereby each component processes at its own rate and completely independently, this approach has advantages. It is much simpler to analyse, more likely to occur, and the sorts of abstract machines typically used in model checking are all closed under lock-step parallel composition. That is, the lock-step composition of two machines of one type is another machines of the same type.

In Chapter 3 we reviewed the literature on parallel decomposition of finite machines, and extended the results therein to ω-machines. The extensions to ω-machines were straightforward and were presented concurrently with the finite machine review. The structural presentation of both finite machines and ω-machines as given in Definition 2.22 and Definition 2.24 were also similar, and the intuitions behind the majority of the results were identical.

We first formalised the concept of parallel composition. The motivation for each definition was that each component would process independently, with a transition in a parallel machine being possible iff it is possible in each component. Acceptance is similar, and the final definition could have been regarded as a Cartesian Product if we view the machines as algebras. Each of the machines considered is closed under

parallel composition.

We then developed precisely the relationship we require to be present between a large machine, and the parallel composition of its "components". We require the components to be smaller, and the (primitive) measure of size that we used was the cardinality of the state space. Equality would have been trivially too specific, similarly for isomorphism. For either, there would be too few large machines that would be decomposable. Thus we utilised the notion of structural inclusion, where the structure of a machine is predominantly carried by its edges.

Structural inclusion is a weak enough relationship to allow that realistic large machines may be decomposable. Unfortunately the weakness of this relationship means that the information garnered from this relationship is slight. In practice, for finite machines at least, the relationship is strengthened by providing some global information to restrict the actions of the parallel machine so that it becomes isomorphic, or at least isomorphic in relevant places, to the original large machine under consideration. These sorts of modifications are essential to the practical application of decompositions, but the primary goal of this work has been to develop the theoretical groundwork upon which these sorts of practical issues may be discussed meaningfully. Because of this, issues relating to the application of the results presented here have not been discussed in great depth.

The concepts of structural inclusion and parallel composition allowed us to state our problem formally—this was done in Definition 3.16. We provided a separate definition for the two component case, and restricted our initial attention to this case because, as shown later in Theorem 3.31, analysis of the two component case suffices.

The necessary and sufficient structural conditions were developed separately. For the necessary conditions the structural properties of the large machine were verified constructively while the intuitions behind the assertions were strengthened by means of a developed example built up as lock-step parallel machine.

The sufficient conditions were developed next. The same example was used as motivation, but the relationship was explored form the opposite direction. Again the proofs were constructive in nature, giving methods to generate parallel decompositions given the appropriate structural information.

When the sufficient conditions matched the necessary ones they were brought together, along with justification that the two component case is sufficient, to give the final result of the chapter. Theorem 3.31 provided an equivalence which means that parallel decompositions are present precisely when SP state partitions are present. The constructive natures of the component proofs actually say something stronger. They say that the study of parallel decompositions is equivalent to the study of SP partitions, and this idea is really the culmination of our finite machine review and ω-machines extension.

We then illustrated how this result enables us to find component machines for a "large" machine which was not built up as a parallel machine. The search for SP partitions leads to an algorithm, presented here only informally, that treats SP partitions as fixed points of a suitable recursion. The "minimal" fixed points uncovered in this way determine if a machine has a parallel decomposition. Minimal here is with respect to the partial order given by member inclusion, defined in Definition 2.5. In fact, the result of Theorem 3.31 could be strengthened to state that a parallel decomposition exists precisely when there are two *minimal* orthogonal non-trivial SP partitions. We choose to present Theorem 3.31 as the culminant result for two reasons. Firstly, it successfully translates the problem of finding parallel decompositions into the problem of finding a suitable property based purely on the algebraic structure, whereas the strengthening of the result only attaches the further equivalence of two structural properties. Additionally, as mentioned above, the presentation shows that parallel decompositions are equivalent to SP partitions. Strengthening the result obscures this natural equivalence, because minimal or prime partitions are not equivalent to decompositions. The equivalence between SP partitions and decompositions is imperative to the application of this theory. It is the maximal partitions which generate the best decompositions (best here meaning that the components are the smallest), and focusing on the minimal ones would distract attention.

All of this provides a coherent development of the ideas central to the next chapter, which is the focus of this work. There, the concepts and intuitions discussed in Chapter 3 were extended to timed machines. The structure of Chapter 4 closely parallels that of Chapter 3 in order that the similarities present between the theories be made transparent. Thus there was a similar developed example threaded throughout the first part of the chapter. Initially the statement of the problem of when a large machine can be expressed as the composition of components was developed. This relied on concepts of parallel composition and structural inclusion similar to those used earlier with finite and ω-machines. Following this, conditions, both necessary and sufficient, were found to demonstrate the equivalence between parallel decompositions and suitable partitions of state and clock spaces. This accomplished the goal of providing a structural means of determining if a machine has a parallel decomposition. Unlike ω-machines, the results for timed machines did not follow directly from the corresponding finite machine results because the inclusion of the clocks allows for different interpretations of the original problem. We chose an interpretation allowing the finite machine results to be generalised without major conceptual leaps. Our definition of parallel composition is such that timed automata are closed under parallel composition.

The work of Chapter 4 extends the decomposition theory to timed machines. The concept of an SP partition is replaced by an admissible pair, consisting of a state partition and a clock subset which are capable of inducing deterministic quotient machines. . The structural property of a timed machine that must be present in order for it to have a parallel decomposition appears to be stricter than the corresponding property for either of the untimed machines. A pair of admissible pairs is required, where

the state partitions are orthogonal and the clock sets are disjoint. Because of this strictness we might expect parallel decompositions to be less common for timed machines. Whether this is true in practice or not remains to be seen. The examples at the end illustrate some of the uses and pitfalls both of the results here, and decomposition in general. It can be useful in reducing the state size considerably, but the definition of decomposition, and hence the results, as stated doesn't ensure that a decomposition will accomplish any reduction in size at all. This is despite the sacrifice of detail that generally accompanies treating a machine as a parallel composition. The greatest reductions arise from the greatest partitions with an appropriate partial order, but finding these is difficult. Indeed, finding suitable state partitions in the first place is difficult, and here we only touch on these difficulties as we progress through worked examples at the end of the chapter.

It appears that the more natural approach to finding admissible pairs (each consisting of a suitable state partition and clock split), and hence parallel decompositions, is to look at what groupings are forced if certain clocks are ignored/removed. This is in contrast to the finite/ω- machine case where the minimal (or prime) partitions are formed by considering what groupings are forced if a pair of states are grouped together.

For timed machines minimal partitions are formed by considering what groupings are forced if a single clock is ignored. If two of the state partitions induced in this manner are not orthogonal, then every admissible pair either contains both the clocks that generated them, or neither. Thus consider the effect of excluding both clocks and repeat the process, building up set of clocks that must be grouped together, and associated state partitions. If this process leads to a trivial state partition then the machine in question has no parallel decomposition. The converse, however, is not valid, and finding a decomposition may require exploring all the possible ways of splitting the clock set into two, and subsequent pairings of states.

One last thing to note again before we close our summary of this chapter is the weakness of the structural inclusion component in the definition of parallel decomposition for each of our types of machines. . This represents the loss of fine detail that arises from treating the large machine as a parallel composition, and means that only certain properties of the original machine carry over to the parallel machine.

5.2 Problems that Arise

The theory discussed here is well intentioned, but to what extent is it applicable to real world problems? In order to address this problem and observe what might be done about it we will group the issues that arise into two broad areas. The first deals with the inherent difficulties that arise in the practical implementation of this and related work, what we call structural difficulties. The second deals with

problems that are specific to the design of an effective algorithm for implementing decompositions. A certain amount of overlap arises.

5.2.1 Structural Problems

The first problems to arise are the ever-present problems involving the representation of a real world system by an abstract or mathematical one. To what extent does the model describe the original, given that the model has abstracted away (ignored) much of the detail present? There is no avoiding this problem, but it serves as good evidence that the application of mathematical tools in general requires some understanding of both the tools and the particular problem, so that intuitions developed in the study of a problem can help guide the choice of tool. Of particular relevance to the theory discussed here are questions relating to the representation of a system by a discrete model.

There are also problems that come about because the developed model carries more information than is necessary to find the answers desired of it. These are also to some extent unavoidable in a generic discussion such as this, being domain and problem specific, and again provide evidence that blind application is unlikely to give rise to useful results. One can imagine that in particular areas heuristics will give some guidance as to what features to ignore.

These issues, whilst pertinent, merely serve as warnings, and offer no solution. Finite machines, or variants, are used extensively throughout the computer science community and beyond, and in a sense can represent any kind of discrete model. Finite machines deal with terminating processes, whilst some variants, for example timed and ω-machines, deal with non-terminating processes. This serves as strong evidence for their use in a range of areas, and motivates a generic study of them. The most prevalent problem that arises in treating a generic system as a finite machine is that for a realistic system, the size of the finite machine is unwieldy, and often too large to represent directly on a modern computer, let alone manipulate. Part of this problem can be alleviated by only concentrating on the parts of the problem that are pertinent. Another way is to modify the structure of finite machines so as to incorporate some salient aspects of the sorts of problems you are looking at. A third approach is to find ways to store a large finite machine which has some structure in a smaller space, for example BDD's [Bry86, And96].

The first of these methods is really a paradigm whose details are problem specific. The second is one of the prime motives for the development of timed machines. So many models, across a large range of fields, involve dealing with time that a very natural way in which to modify the definition of a finite machine is to incorporate some sorts of time dependence. Many approaches have been considered, and we have chosen to use the one presented in [AD94], because it seems to be the most popular at present. The third approach is the one that leads to the development of the decomposition theory presented here.

Most of the time finite machines are used, they are built to describe a model, and then analysed in some manner to obtain information about the original system. The analysis typically requires repeated random access of the machine, and as such, conventional compressions are of limited use. Thus we try to take advantage of any structure present in the machine in other ways. The lock-step parallel compositions we consider here are one such way. The disadvantage of these is that they require a great deal of structure to be present in the machine. Very few large machines are expressible as the parallel composition of two smaller machines. To avoid this problem, we only require structural inclusion. Thus a machine is decomposable if it is contained in the parallel composition of two smaller machines. Not entirely unexpectedly, this raises other problems. Most notably, requiring merely containment means that the amount of information that can be generated about the original machine is reduced significantly. For example a property of the form "For all runs, ..." can be verified if the parallel machine has the required property, but cannot be refuted even when the parallel machine does not have the property—it may be that only the runs that are present in the parallel machine and not the original are serving as counterexamples. One situation where this problem has arisen is in the use of finite machines in model checking. For example, if we have a property ϕ, a program (represented by a finite machine) P, and we wish to check if P satisfies ϕ. The generic algorithm for verifying this checks to see if each word generated by P is a word that satisfies ϕ, translating it into a language containment problem. If P was decomposed, say in to P_1 and P_2, then checking to see if $P_1 \| P_2$ satisfies ϕ can only give some information. More specifically if the test is positive, that is $P_1 \| P_2$ does satisfy ϕ, then P must also satisfy ϕ. Negative tests, however, give no information. Just as a side note, the results of Lemma 3.7 and Lemma 4.5 show that one way of verifying that $P_1 \| P_2$, and hence P, satisfies ϕ is to verify that both P_1 and P_2 satisfy ϕ individually.

The limitations of homomorphisms are a significant problem, and there have been numerous attempts to deal with it. A common method is to provide additional information detailing which transitions are present in the parallel machine and not in the original. To be of benefit this method requires that the number of additional transitions is not too large, and so requires more structure to be present in the original machine, and also that the decomposition chosen reflects this. Finding these appropriate decompositions can be hard. An additional problem arising using the decomposition results here is that the only requirement on a decomposition is that both components are smaller than the original machine. In practice it is (comparatively) easy to find decompositions so that the components are smaller using the method of partitions discussed here. Invariably, these easy-to-find partitions give rise to components which are only slightly smaller than the original machine meaning that the total storage required for the two components is probably more than that for the original. The smallest components require finding the largest partitions, and this can be very hard (worst case scenarios require checking all partitions).

5.2.2 Algorithmic problems

Another problem, one that on the face of it appears to be more tractable, is that the theory as presented here assumes knowledge of the entire machine beforehand, and thus potentially requires analysis of an impractically large machine in order to represent it as the composition of smaller ones. This is precisely the problem we wished to avoid in the first instance. To combat this would involve building up the parallel decompositions as the machine is developed, rather than waiting until the whole structure is present. A similarly undisparaging problem is that often finite machines have a very sparse transition structure, which is lost when represented as components. Often there are actions which do not affect the state of a component. Typically, these actions are ignored—indeed, this convention has been observed here—and this results in a slightly more general notion of parallel composition. This notion has a distributed alphabet between components [Zie87] to remove redundancy in the action set. This doesn't broaden the range of finite machines that can be theoretically decomposed, but it does broaden the range of machines that can be decomposed in practice, and also speeds up the resulting analysis.

The preceding discussion was largely based on the application of the theory to finite machines, because the theory for finite machines has been around long enough that these problems have arisen. Most of the problems that arise are likely to be common to ω-machines and timed machines as well, and although solutions have not been developed for them yet, the finite machine case should again serve as a guide. Timed automata in particular appear to be of the most potential use, as the size and complexity reductions coming about by developing models with time rather than discretizing and treating as finite machines is considerable. Timed machines do still suffer from large state spaces, although not to the same extent as finite machines.

The primary advantage of the abstract machine theory discussed here is that it is purely structural and hence general. The main primary disadvantage is that there is no user guidance. Finding decompositions would be much more practical if users were able to guide the algorithm to look for particular partitions, based on their knowledge of what the machines are modelling. It may also be possible to restrict the sorts of decompositions so that the parallel machine will not accept certain critical strings, very much as the non-conflicting controller problem.

The problems restrict the areas where this theory can be applied at present, but not its use in those areas. Any resolution of these issues will serve to broaden considerably the areas where abstract machine theory can be applied.

5.3 Possible Extensions

Although the theory looked at here is by no means complete, it might serve as a useful note to see how the theory as it stands works when applied to existing tools. It may be that even as it stands, the concepts of parallel decomposition may result in faster methods. Binary decision diagrams (BDD's) are common tools for representing large finite machines, particularly in model checking, see for example [And96], and there are many tools available that make use of them. It would be a good idea to see how the parallel decomposition by partitions methods work with BDD's. There are also possibilities for designing simpler hardware circuits and the automatic synthesis of controllers [EA95], although more pronounced benefits are likely if the theory is extended. Even if the theory as it stands is of no direct benefit in these areas, there are numerous ways to modify the concepts, and some of them may be of more use.

As an illustration consider the problem of controller synthesis. There are numerous ways that the sorts of concepts looked at here can be extended, and for finite machines there is already work in this area. For example, often a sythesised controller will be very large, and perhaps best treated as components. Another possibility is that the plant will be described as a parallel machine, and it would be desirable to express the controller similarly, where it is required that the parallel representation of the controller is itself a controller. Such controllers are called non-conflicting [RW89].

There are also many sources of possible extensions arising from dealing with the more generic problems of Section 5.2. Several possibilities seem especially tempting.

The first is to develop more problem specific types of machines, which take advantage of the particular structures present in problem domain. This was part of the original motive behind the development of timed machines [AD94], and they have been very successful in achieving this goal. A possible example might be to develop variants of timed machines whose clocks may progress at varying rates. The foremost advantages of these kinds of extensions are that considerably less storage will be required, and the models might be more intuitive to build up. The follow on advantages include the fact that analysis will more likely be feasible, quicker and also more transparent to the user/designer. The two foremost disadvantages are that the machines will only be of benefit on a reduced range of problems, and that the theory developed here would have to be reworked for each type of machine. Another possibility is to restrict the forms of the machines to obtain stronger results. There is a lot of work in this area, where a restricted form of timed machine gives a stronger (and hence more useful) decomposition result. An example is [KM], where a restricted form of clock/constraint language is considered. Each clock can only be reset on a particular action, and the constraint language is highly restricted because the aim is the modelling of sensors. Likewise the concepts of homomorphism are adapted, and the resulting theory is much stronger.

This last issue leads to another area of possible work. Treating the theories developed here as instances of a more general theory. Certain topics in Universal Algebra such as the theory of ω-algebra's appears to be applicable here. But in order for the applicability of the theory to be recognised, the results should be presented along with the particular instances of their use. It is unfortunate that at present the theory is abstract in more than one use of the word. This approach could lead to the largest benefits to the designers of models, as it may be suitable for them to develop more problem specific sorts of machines, and the intuitions built up by studying the general theory developed from instances could be applied to the machines they develop, thus producing a theory that is broad in applicability whilst still being of benefit.

Another approach is to consider other sorts of compositions/decompositions, for example the wreath product decomposition of Krohn-Rhodes discussed in [Hol82]. After all, a decomposition is really little more than a smaller representation of a machine, and as such relies on the fact that many machines have the sorts of structure that the decomposition can take advantage of. The parallel composition presented here takes the form of a lock-step parallel decomposition which is useful, but in effect this forces communication between components to the extent that each must progress at the same rate. Allowing for different components to process inputs at different rates, and/or allowing for more explicit communication between components could again make the decompositions more applicable, and also more intuitive. A potential cause for concern is that under the more intuitive concepts of composition the machines we have dealt with here might not be closed. Thus this approach may require the sorts of machines under consideration to be expanded. This work would be of particular interest to the development of methods of simpler implementation for hardware circuits, where continual checking against a global clock is infeasible.

As an instance of the last of these approaches, one idea which really came about by considering the second approach is changing the concept of a parallel decomposition of timed machines so that it is not states and clocks which are mapped, but extended states. This is an inclusion of a derived structure rather than merely taking the structure as given. The main reason for considering this is that timed determinism requires only one transition for each *extended state*, rather than for each state or clock. The decomposition thus generated will match decompositions of a timed machine with suitable partitions of the extended state space. A more useful way of viewing this then is as a suitable partition on equivalence classes of clocks and states, in other words of the region machine. When discussing the motivation behind the decomposition theory earlier in Chapter 4 it was noted that the partitions thus generated are ignoring some potential sources of information, notably that if some states are joined, and a transition out of either is considered, then more than just the new state is known. It is also known that the clocks must have

satisfied the constraints on the appropriate edges. In the theory presented in Chapter 4, this information was ignored, and the above corresponds to not ignoring it.

As mentioned earlier, this list of possible extensions is by no means exhaustive, but is included to indicate that this area of research is still in its infancy, whilst having the potential to be of great use in a number of areas, and as such deserves attention.

Bibliography

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[And96] Henrik Reif Andresen. An introduction to binary decision diagrams. September 1996. Course Notes for C4340 E96, Department of Computer Science, Technical University of Denmark.

[Arb69] Michael A. Arbib. *Theories of Abstract Automata.* Prentice-Hall, Inc., 1969.

[BCMH92] J. R. Burch, E. M. Clarke, K. L. McMillan, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98:142–170, 1992.

[BDG88] José Luis Balcázar, Josep Diaz, and Joaquim Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

[Bry86] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C35(8):677–691, 1986.

[BS81] Stanley Burris and H.P. Sankappnavar. *A course in Universal Algebra.* Springer-Verlag, 1981.

[Cod68] E.F. Codd. *Cellular Automata.* Academic Press, Inc., 1968.

[Coh65] P. M. Cohn. *Universal Algebra.* Harper and Row Ltd. and John Weatherhill, Inc., 1965.

[EA95] A.Pnueli E. Asarin, O. Maler. Symbolic controller systems for discrete and timed systems. *Lecture Notes in Computer Science - Hybrid Systems II*, 999:1–20, 1995.

[Eme90] E. A. Emerson. Automata on infinite objects. In J. van Leeuwen, editor, *Temporal and Modal Logic*, pages 995–1072. Elsevier, 1990.

[Gin68] Abraham Ginzburg. *Algebraic Theory of Automata.* Academic Press, Inc., 1968.

[GP72] F. Gécseg and I. Peák. *Algebraic Theory of Automata.* Number 2 in Disquisitiones Mathematicae Hungaricae. Akadémiai Kiadó, 1972.

[Gra68] George Graätzer. *Universal Algebra.* The University Series in Higher Mathematics. D. Van Nostrand, 1968.

[Hal60] Paul R. Halmos. *Naive Set Theory.* D. Van Nostrand Company, Inc., 1960.

[Hol82] W. M. L. Holcombe. *Algebraic Automata Theory.* New York : Cambridge University Press, 1982. Cambridge studies in advanced mathematics 1.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979. in 3 hour loan.

[Jac80] Nathan Jacobsen. *Basic Algebra II.* W.H. Freeman and Company, 1980.

[Kel95] Dean Kelley. *Automata and Formal Languages.* Prentice Hall, Inc., 1995. An introduction.

[KM] Padmanabhan Krishnan and Kahn Mason. A theory of decompostion for a subclass of timed automata. *Submitted.*

[Kri] Padmanabhan Krishnan. Decomposing controllers into non-conflicting distributed controllers. Private Communication.

[Lap78] E.S. Lapin. *Semigroups.* American Mathematical Society, 1978.

[LS91] F. William Lawvere and Stephen H. Schanuel. *Conceptual Mathematics.* Buffalo Workshop Press, 1991. An introduction to Category Theory.

[MP94] Oded Maler and Amir Pnueli. On the Cascaded Decomposition of Automata, its Complexity and its Application to Logic. *unpublished*, 1994.

[RW89] P. J. G. Ramadge and M. Wonham W. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

[Shi87] M. W. Shields. *An Introduction to Automata Theory.* Blackwell Scientific Publications, 1987. Computer Science Texts.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics.* Elsevier, 1990.

[Var96] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure ver sus Automata*, volume 1043 of LNCS, pages 238–266. Springer-Verlag, 1996.

[Wec92] W. Wechler. *Universal Algebra for Computer Scientists.* Springer-Verlag, 1992.

[Whi98] Alfred North Whitehead. *A Treatise on Universla Algebra with applications.* Number 1. Cambridge University Press, 1898.

[WN94] Glynn Winskel and Mogens Neilsen. *Models for Concurrency.* BRICS, Dept. of Computer Science, Uni. of Aarhus, 1994.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *Theoretical Informatics and Applications*, 21(2):99–135, 1987.

[ZM95] Ying Zhang and Alan K. Mackworth. Synthesis of hybrid constraint-based controllers. *Lecture Notes in Computer Science - Hybrid Systems II*, 999:552–567, 1995.